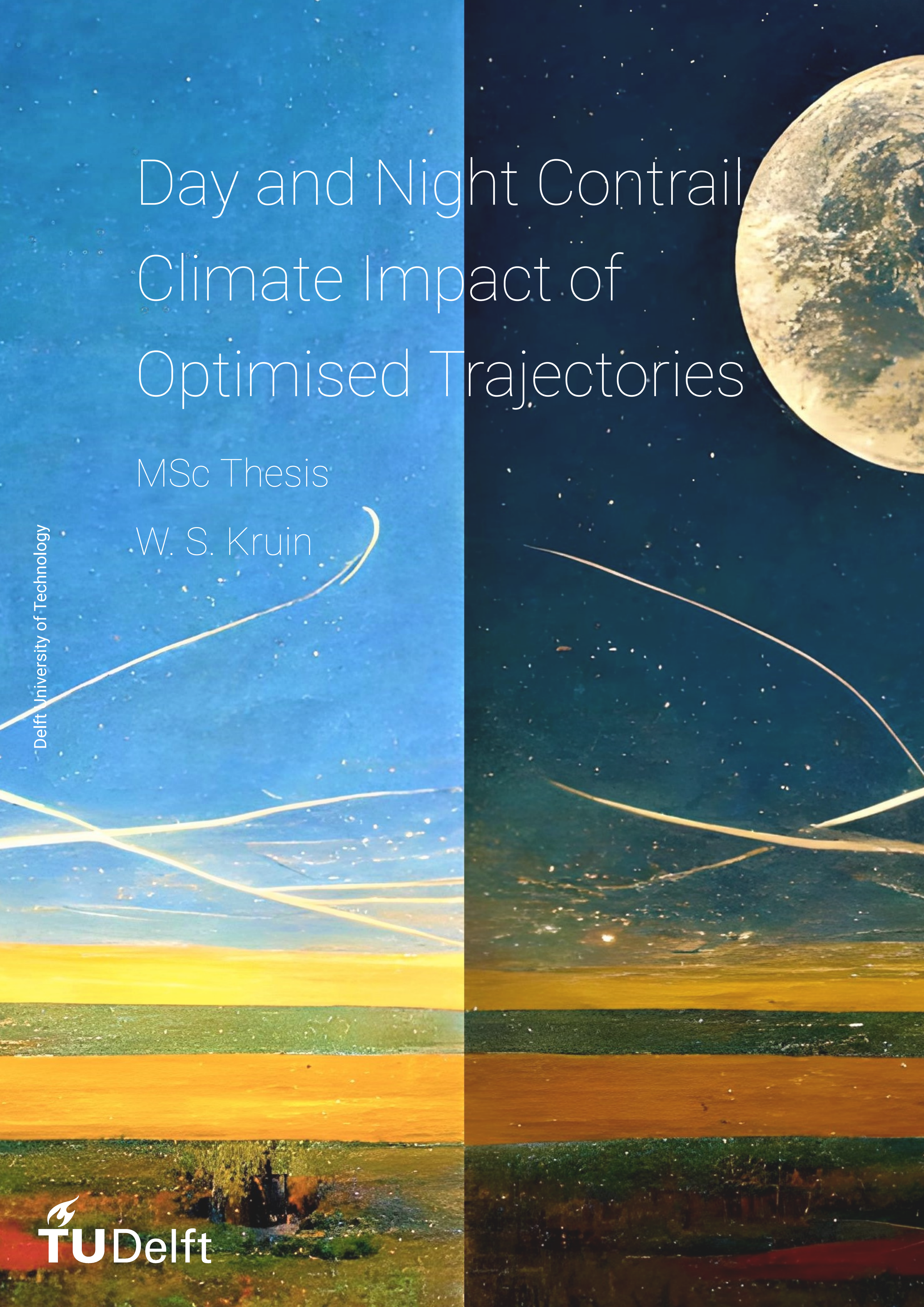

Day and Night Contrail Climate Impact of Optimised Trajectories

MSc Thesis

W. S. Kruin

Delft University of Technology

This page has intentionally been left blank. The cover picture is generated by the author using Midjourney (`https://midjourney.com/`).

Day and Night Contrail Climate Impact of Optimised Trajectories

by

W. S. Kruin

to obtain the degree of Master of Science
at Delft University of Technology,
to be defended publicly on December 8th, 2022 at 10:00.

Student number:	4672771		
Project duration:	October 28th, 2021 - December 8th, 2022		
Thesis committee:	Chairman:	Prof.dr. V. Grewe	TU Delft | DLR
	Supervisor:	Dr. F. Yin	TU Delft
	Supervisor:	F. Castino, M.Sc.	TU Delft
	External member:	Dr.ir. G. La Rocca	TU Delft

Preface

In front of you lies the final piece of the puzzle that is called ‘a master degree in aerospace engineering’. And what a puzzle it was! Certainly it was a challenging one to complete, demanding hard work, long days, short nights, dedication and determination. Due to the COVID-19 pandemic, parts of the puzzle had to be made from home, while other parts of the puzzle were put together in a hybrid fashion. However, despite (but maybe also thanks to) the challenging nature of this puzzle, completing it turned out to be one of the most fun and insightful periods in my life.

When I started the bachelor aerospace engineering in Delft, I felt the need to defend my choice to become part of an industry that is infamous for its negative environmental impact. I ensured my friends and family that I would only join forces with ‘these people’ to eventually make aviation more sustainable. Although sincere, it did not take long for this promise to become a faint idea resting somewhere in the back of my head. Honestly, it took until the start of my M.Sc. degree to remember my pledge, when it was time to select the electives that allow a student to dive deeper into a subject of his own choice. Courses like Aircraft Noise and Emissions and Aircraft Emissions and Climate effects immediately caught my attention, and I could not regret my choice for these subjects less, as they eventually became the cradle for my thesis subject.

I would not have been writing this preface without the support of some amazing people for who I am very grateful. Thank you, friends in Delft. Throughout the the last five years you made every day’s journey from Haarlem to Delft worthwhile and I am sure our friendships will last well beyond Delft. Thank you, mom, dad and Thandi, for being a warm family. I could not have done it without your support. Lydia, few students get to know the joy of having the everlasting support of a spouse during their thesis. Even fewer know the blessing of that spouse to be you. Thank you for always being there, regardless my highs and lows. I am thankful for the ANCE group for warmly taking up this FPP’er. A special thanks goes out to Federica, who helped me find and convey both the big and detailed picture of my thesis during numerous Tuesday mornings and more. Thank you, Feijia, for our fruitful discussions, thesis related, but also career related. I would also like to thank Volker for his helpful feedback at every milestone that led to this thesis.

Calling this thesis ‘the final piece’ might be true for my journey as a student, but I feel like this piece is only the beginning of a new, bigger and even more exiting puzzle. Luckily, I am not done puzzling yet.

Wessel Kruin
Haarlem, November 2022

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

List of symbols.

Symbol	Parameter	Unit
a	Tuner for the calculation of r_{co}	$[-]$
$aCCF_{contrail}$	Contrail aCCF	$[K/km(contrail)]$
A_{eff}	Effective albedo of the Earth-atmosphere system	$[-]$
$ATR20_{contrail}$	Contrail ATR20	$[K]$
$\overline{ATR20}_{contrail}$	Specific contrail ATR20	$[K/km]$
$ATR20_{total}$	Total ATR20	$[K]$
b_{ci}	Natural cloud coverage	$[-]$
b_{co+ci}	Maximum possible cloud coverage	$[-]$
c_f	Unit fuel costs	$[\$/kg]$
c_p	Air specific heat capacity	$[kJ/kgK]$
c_t	Unit time costs	$[\$/s]$
d	Flight distance	$[m]$
d_e	Effective flight distance	$[m]$
D	Contrail thickness	$[m]$
$\overline{eATR20}_{contrail}$	Effective specific contrail ATR20	$[K/km]$
$\overline{eSOC}$	Effective specific simple operating cost	$[\$/km]$
EI_{H_2O}	Emission Index of H_2O	$[kg_{H2O}/kg_{fuel}]$
E_{LW}	Long-wave extinction factor accounting for a cirrus above a contrail	$[-]$
E_{SW}	Short-wave extinction factor accounting for a cirrus above a contrail	$[-]$
ERF	Effective Radiative Forcing	$[W/m^2]$
$FlightDist$	Flight distance	$[km]$
F_{cr}	Fuel flow	$[kg/s]$
$FUEL$	Fuel use	$[kg]$
G	Slope of mixing line	$[Pa/K]$
h	Flight altitude	$[m]$
IWC	Ice Water Content	$[kg/m^3]$
k_T	Model parameter of Schumann et al. (2012)	$[W/m^2K]$
$M(air)$	Air molar mass	$[g/mol]$
$M(H_2O)$	Water molar mass	$[g/mol]$
OLR	Outgoing Long-wave Radiation	$[W/m^2]$
p	Pressure	$[Pa]$
p^{H_2O}	Water vapour partial pressure	$[Pa]$
PCC	Potential Contrail Coverage	$[-]$
$PCCDist$	Contrail distance	$[km]$
PCC_{Dist}	Contrail distance	$[km]$
$Potcov$	Potential persistent contrail cirrus coverage	$[-]$
Q	Fuel specific heat content	$[J/kg]$
r	Grid mean relative humidity	$[-]$
r_{ci}	Critical relative humidity for cloud formation	$[-]$

List of symbols (continued).

Symbol	Parameter	Unit
r_{co}	Critical relative humidity for partial grid box ice-supersaturation	$[-]$
r_{eff}	Particle distribution effective radius	$[\mu m]$
rh	Relative humidity above water	$[-]$
rhi	Relative humidity above ice	$[-]$
r_{nuc}	Relative humidity of homogeneous freezing threshold	$[-]$
r_{SAC}	Relative humidity where SAC is satisfied	$[-]$
r_{sat}	Relative humidity at saturation	$[-]$
r^*	Threshold relative humidity for $b_{co+ci} = 1$	$[-]$
RF	Radiative Forcing	$[W/m^2]$
RF_{LW}	Long-Wave Radiative Forcing	$[W/m^2]$
RF_{SW}	Short-Wave Radiative Forcing	$[W/m^2]$
SDR	Sun Direct Radiation	$[W/m^2]$
SOC	Simple operating cost	[\$]
$\overline{SOC}$	Specific simple operating cost	$[\$/km]$
S_0	Solar constant	$[W/m^2]$
t	Time	$[s]$
t_{life}	Contrail lifetime	$[h]$
$t_{sunrise}$	Time until sunrise	$[h]$
t_{sunset}	Time until sunset	$[h]$
t_A	Model parameter of Schumann et al. (2012)	$[-]$
t_0	Reference time	$[s]$
T	Temperature	$[K]$
$TH_{sunrise}$	Threshold time before sunrise for day or night contrail aCCF	$[h]$
TH_{sunset}	Threshold time before sunset for day or night contrail aCCF	$[h]$
T_0	Model parameter of Schumann et al. (2012)	$[K]$
u_1	Model parameter of Unterstrasser and Gierens (2010)	$[m^2/kg]$
u_2	Model parameter of Unterstrasser and Gierens (2010)	$[m^3/kg]$
w_{day}	Daytime contrail aCCF weight factor	$[-]$
w_{night}	Nighttime contrail aCCF weight factor	$[-]$
X	Vector of AirTraf design variables	$[coordinate]$
α_c	Contrail albedo	$[-]$
β_{ext}	Extinction coefficient	$[m^{-1}]$
ΔT	Temperature response	$[K]$
$\Delta\lambda_{airport}$	Longitudinal distance of airport pair	$[deg]$
η	Propulsive efficiency	$[-]$
θ	Solar zenith angle	$[deg]$
λ	longitude	$[deg]$
λ	Climate sensitivity parameter	$[K/(W/m^2)]$
μ	Cosine of solar zenith angle	$[-]$
τ	Contrail optical thickness	$[-]$
τ_c	Optical depth of cirrus above a contrail	$[-]$
ϕ	Latitude	$[deg]$

Symbol subscripts.

Symbol	Refers to
d	day number
f	flight number
i	Segment number / waypoint number / solution number
j	Jet
m	mid layer
s	Saturation with respect to water
si	Saturation with respect to ice
tot	Total
∞	Ambient condition

List of abbreviations.

Abbreviation	Spelled out fully
aCCF	Algorithmic Climate Change Function
AGWP	Absolute Global Warming Potential
ARMOGA	Adaptive Range Multi-objective Genetic Algorithm
Arr	Arrival
ATM	Air Traffic Management
ATM4E	ATM for the benefit of environment and climate
ATR20	Average Temperature Response over a 20 year time horizon
BADA	Base of Aircraft Data
CCF	Climate Change Function
CO_2	Carbon dioxide
CP	Control Point
Dep	Departure
DLR	Deutsches Zentrum für Luft- und Raumfahrt
ECHAM	European Centre Hamburg general circulation model
ECMWF	European Centre for Medium-Range Weather Forecasts
EMAC	ECHAM/MESSy Atmospheric Chemistry
ETO	Estimated Time Over
F	Future emission scenario
FlyATM4E	Flying ATM4E
FL	Flight Level
GC	Great Circle
H_2O	Water
ICAO	International Civil Aviation Organisation
IPCC	Intergovernmental Panel on Climate Change
LW	Long-Wave
MESSy	Modular Earth Submodel System
NO_x	Nitrogen oxides
P	Pulse emission scenario
REACT4C	Reducing Emissions from Aviation by Changing Trajectories for the benefit of Climate
SAC	Schmidt-Appleman Criterion
SOC	Simple Operating Costs
TOM	Trajectory Optimisation Module

List of abbreviations (continued).

Abbreviation	Spelled out fully
SW	Short-Wave
TOA	Top Of Atmosphere
UTC	Coordinated Universal Time
3D	3 Dimensional
4D	4 Dimensional

Abstract

Contrail formation is one the largest warming contributions of aviation's climate impact. Measures to mitigate contrail climate impact include the optimisation of flight trajectories to avoid the formation of warming contrails or to find air space where extra cooling contrails are formed. The latter option is only possible during daytime, due to the interaction of a contrail with the short-wave radiation from the Sun. Little research is done to evaluate the effect of the diurnal cycle on contrail climate impact mitigation. Moreover, models to predict this the contrail climate impact are still in the development face. This thesis aims to (1) find the daytime and nighttime contrail climate impact of eco-efficient flight trajectories, (2) to assess the prediction's robustness and (3) to recommend a best practice for eco-efficient flying by using the difference between the daytime and nighttime contrail climate impact. To this end, simulations were carried out within the EMAC climate chemistry model, aided by the submodels AirTraf, CONTRAIL and ACCF. The robustness of the results were tested against a model parameter: the threshold time until sunrise ($TH_{sunrise}$). When a contrail is formed during nighttime, $TH_{sunrise}$ determines whether the contrail is formed sufficiently close to sunrise to be considered a daytime contrail. It is concluded that winter daytime mitigation of contrail climate impact is more eco-efficient than winter nighttime mitigation. Overall, daytime mitigation achieves a larger maximum reduction of contrail climate impact than nighttime mitigation. Moreover, summer mitigation is more effective than winter mitigation and winter mitigation has a larger reliance on the formation of extra cooling contrails. The thesis results are robust with respect to a varying $TH_{sunrise}$. However, overall results become biased to daytime results or nighttime results. Lastly, it is shown that differences between day and night contrail climate impact mitigation allow for the enhancement of mitigation results.

Chapter 1

Introduction

1.1 Background

With their sixth and latest assessment report the Intergovernmental Panel on Climate Change (IPCC) has made it clear: human influence has unequivocally warmed Earth's atmosphere, ocean and land (IPCC (2021)). The rise in global temperature *"has caused widespread adverse impacts and related losses and damages to nature and people"* (IPCC (2022)). To what extent ecosystems and humans will face additional risks depends on our ability to limit further warming of our planet (IPCC (2022)). The IPCC report underlines how important it is to live up to the promise many countries made by signing the Paris agreement in 2015: to pursue efforts to keep global warming below 1.5 degrees Celsius with respect to the pre-industrial level. Keeping this promise means that action needs to be taken to reduce green house gas emissions and effects across all sectors. One of the major contributors to anthropogenic climate change is the aviation sector. According to Lee et al. (2021), air traffic accounted for 3.5% of the net anthropogenic Effective Radiative Forcing (ERF) in 2011. Given that the amount of ERF induced by the aviation sector is projected to grow even further (Lee et al. (2021)), it is a great challenge to mitigate the climate impact of air traffic.

Whereas most people associate global warming with the emission of CO_2, it is actually the collective of non-CO_2 effects that make up the largest part of the climate impact of air traffic. Nearly 2/3 of the net aviation ERF finds its origin in the emission of NO_x, water vapour and aerosols or the formation of contrail cirrus (Lee et al. (2021)). With some uncertainty, it is this last term that Lee et al. (2021) identified as the single largest warming term contributing to aviation's net ERF. Given this knowledge, the mitigation of the climate impact of contrails is an interesting focus area to reduce aviation's overall climate footprint.

Means by which the climate impact of contrails could be mitigated can be divided into two categories: technological measures and operational measures. An example from the first category is the onboard storage of water vapour that would otherwise be emitted and become a condensation trail. The hot water vapour is then cooled before emission such that the vapour condensates on board and can be stored in water tanks (Noppel and Singh (2007)). Additionally, new fuels like hydrogen and synthetic kerosene are entering the aviation market. These fuels are characterised by lower soot numbers than conventional kerosene, resulting in contrails with a shorter lifetime and a lower climate impact (Narciso and de Sousa (2021), Bräuer et al. (2021)).

Measures in the second, operational, category do not involve changes to the aircraft or the fuel that it burns, but rather refer to changes in Air Traffic Management (ATM) operations. A vast collection of studies shows that the climate impact of contrails (and that of other non-CO_2 climate forcers) is highly dependent on the location of formation, the local meteorological conditions and the time of day (e.g., Meerkötter et al. (1999), Fichter et al. (2005), Kärcher (2018)). This spatiotemporal variability of non-CO_2 climate impact implies that the trajectory of a flight can be optimised to minimise its resulting climate impact (Grewe et al. (2014a), Grewe et al. (2017), Matthes et al. (2021)). Following up on this possibility, many efforts are made to investigate the opportunities of such trajectory optimisations. Examples are the study into altitude changes by Matthes et al. (2021), the inclusion of a climate optimal routing strategy in a 4D trajectory optimisation model by Yamashita et al. (2016) and the study of Newinger and

Burkhardt (2012) into the effect of daytime restricted flight. Several preliminary small scale tests also show promising results, indicating the viability of climate optimised flight trajectories [1]. Still, many open questions remain on the quantification of the climate impact of a flight and on how to use this information to come to a climate optimal trajectory.

1.2 Objective

Part of the latest research efforts to answer questions on the operational climate impact mitigation of air traffic can be found within the ATM4E and FlyATM4E projects (ATM4E (2018), FlyATM4E (2020)). The ATM4E (ATM for the benefit of Environment and climate) project, which finished in April 2018, explored the feasibility of a concept for the assessment of the climate impact of ATM operations. This concept is based on a set of algorithmic Climate Change Functions (aCCFs) (Yin et al. (2022)). The aCCFs are used to quantify the climate change of a unit emission or unit contrail distance in terms of the average temperature response over a 20 year time horizon (ATR20). Together, the aCCFs form a computationally efficient tool that can determine the overall climate impact of a trajectory and any possible alternatives.

To build further on the findings of ATM4E, FlyATM4E (Flying ATM4E) was started in June 2020. The FlyATM4E project aims to improve ATM4E's assessment method and has the objective to achieve optimised trajectories that feature a robust climate impact reduction. The focus of FlyATM4E's research is on those trajectories that achieve substantial climate impact reduction at a low amount of extra costs, so-called "eco-efficient solutions". The thesis presented in this document is found in the context of the FlyATM4E project and aims to contribute to its objectives.

A literature study, which has been carried out in an earlier stage, showed how this thesis can contribute to the objectives of FlyATM4E. The study of Meerkötter et al. (1999) showed that the local time of day is an important factor towards the climate impact of a contrail cirrus. During nighttime, a contrail serves as a blanket over the Earth and traps outgoing long-wave radiation from the Earth's surface, resulting in a warming effect. However, during daytime, additional to the interaction with long-wave radiation, short-wave radiation from the Sun is partly reflected by the contrail cirrus, which may result in a net cooling effect (Burkhardt and Kärcher (2011), Schumann et al. (2017)). That the interaction between contrails and radiation differs between day and night results in a different contrail climate impact for daytime and nighttime. In turn this difference also imposes an effect on efforts to mitigate the net contrail climate impact of air traffic. For example, both during day and night the avoidance of warming contrails is a way to reduce the net contrail ATR20, but only during daytime flight trajectories may also be adjusted to deliberately create cooling contrails. Therefore, it is interesting to study if this results in an advantage for the daytime mitigation of contrail ATR20. Moreover, the ethical appropriateness of the deliberate creation of cooling contrails is debatable, as it is an active interference in the Earth's climate system. This leads to the question how the results of mitigation efforts are affected if we choose not to employ extra cooling contrails. Furthermore, a relevant question is whether we can benefit from differences between daytime and nighttime contrail ATR20 mitigation, to enhance the results of trajectory optimisation.

Additionally, considering the contrast between daytime and nighttime contrails, the set up of the contrail aCCF imposes an uncertainty on its forthcoming results. Acknowledging the difference between daytime and nighttime contrails, the contrail aCCF uses a different function for daytime and nighttime. Whether a contrail is treated as daytime or nighttime contrail by the aCCF could be determined by a simple question: is it day or night at the time of formation? However, knowing that a contrail may persist for several hours (Vazquez-Navarro et al. (2015)), it may be more sensible to inspect whether the largest part of the contrail lifetime is during night or day. Alternative criteria may also be considered. Whichever criterion is chosen, the

[1] `https://satavia.com/news/`, accessed: November 29, 2022.

chosen approach to distinguish between a daytime and a nighttime contrail may affect the outcome of a trajectory optimisation based on the aCCFs. Hence, the corresponding variability of the results needs to be evaluated to properly value the predicted potential to mitigate climate impact.

As a result of the literature study, the following research goal is formulated:

> *To find the daytime and nighttime contrail climate impact of eco-efficient flight trajectories, to assess the prediction's robustness and to recommend a best practice for eco-efficient flying by using the difference between the daytime and nighttime contrail climate impact.*

Corresponding to this goal, the following research objectives are identified:

1. *To analyse the daytime and nighttime difference in contrail ATR20 for a set of eco-efficient trajectories.*
2. *To assess the variability of the contrail ATR20 of eco-efficient trajectories when considering a change in threshold time until sunrise for nighttime contrails.*
3. *To explore a climate optimal practice to spend a certain budget of simple operating costs for trajectory optimisation on the minimisation of contrail ATR20.*

1.3 Structure

Chapter 2 will visit relevant literature, supporting why trajectory optimisation for the mitigation of contrail climate impact is viable and how this is done. Chapter 3 describes and motivates the methodology applied in this thesis. Results are presented in Chapter 4, after which the verification and validation of these results are discussed in Chapter 5. Chapter6 discusses the meaning and relevance of the results. Lastly, Chapter 7 concludes this thesis and provides recommendations for future work.

Chapter 2

Operational Contrail Climate Impact Mitigation

The concept of eco-efficient trajectories relies on the spatiotemporal variability of air traffic's total climate impact. For trajectory optimisation only aiming for the mitigation of contrail climate impact, this dependency reduces to solely the spatiotemporal variability of contrail climate impact. Firstly, this chapter describes the main ambient parameters that drive contrail climate impact and how these parameters cause its spatiotemporal variability. Secondly, this chapter visits how this variability is employed to come to optimised trajectories.

2.1 Spatiotemporal Impact Variability

To explain why the contrail climate impact of an aircraft varies with location and time, we refer to the growing body of literature that discusses contrails and their Radiative Forcing (RF). Although the first sightings of contrails date back to World War One (Weickmann (1919), Varney (1921)), it was only at the start of the Second World War that the why and how of contrails gained in scientific interest. Heiërman (1944), Brewer (1946), Descamps (1945), Appleman (1953), Scorer (1955) and Scorer and Davenport (1970) explained the origin of contrails as generally accepted nowadays: contrails form due to the condensation of water vapour on ambient or emitted nuclei after which the water freezes. A set of ambient and aircraft variables determines whether a contrail can indeed form, how long it can persists and what will become the value of the resulting RF. Given a certain aircraft, ambient variables are responsible for the spatiotemporal variability of the contrail climate impact throughout the airspace. The sections below will highlight the most relevant relations between contrail climate impact and these ambient parameters.

2.1.1 Schmidt-Appleman Criterion

The thermodynamic theory for contrail formation was developed by Schmidt (1941) and later Appleman (1953). An important aspect within the theory of Schmidt and Appleman is the concept of mixing, that is, the mixing of the engine exhaust gas with ambient air. The combustion of kerosene (and alternative fuels like synthetic kerosene and hydrogen) produces water (H_2O) vapour. The amount of released H_2O is often expressed by the water vapour Emission Index (EI_{H_2O}), in units of kg_{H_2O}/kg_{fuel}. A typical value for the EI_{H_2O} of kerosene is 1.25 kg_{H_2O}/kg_{fuel} (Schumann (1996)), although the precise value varies with exact fuel composition and combustion process. Together with the expelled H_2O, the aircraft engine releases heat into the wake of the aircraft. The amount of released heat depends on the fuel specific heat content, Q [J/kg], and the overall propulsive efficiency of the aircraft, η [-]. While the exhaust air is thus hot and humid, the ambient air is cold and dry, especially at cruising altitudes. Immediately upon contact with each other, the exhaust gas and ambient air undergo a mixing process, during which the exhaust wake reduces both in temperature and humidity.

Whether the water vapour within humid air condensates, depends on temperature (T [K]) and the water vapour partial pressure (p^{H_2O} [Pa]). For a given temperature, the p^{H_2O} at which the rate of condensation equals the rate of evaporation is called the water saturation partial pressure (p_s^{H2O} [Pa]). When $p^{H_2O} = p_s^{H2O}$, a thermodynamic equilibrium is reached at the surface of the liquid. If the p^{H_2O} of air is above p_s^{H2O}, the condensation rate is larger than the rate of evaporation. The air is then called supersaturated and features a growth in liquid water volume. Vice versa, air with p^{H2O} below p_s^{H2O} is called subsaturated and induces a volume shrink of any existing water droplets. In a similar way, the rate of deposition and sublimation of ice is driven by the p^{H2O} and the ice saturation partial pressure (p_{si}^{H2O} [Pa]). For a range of temperatures, p_s^{H2O} and p_{si}^{H2O} were experimentally determined by Sonntag (1994) and others. The partial pressure is often expressed relative to one of the two saturation partial pressures as the relative humidity above water (*rh* [-]) or the relative humidity above ice (*rhi* [-]). These values are computed with equations 2.1 and 2.2.

$$rh = \frac{p^{H_2O}}{p_s^{H2O}} \tag{2.1}$$

$$rhi = \frac{p^{H_2O}}{p_{si}^{H2O}} \tag{2.2}$$

Schmidt and Appleman concluded that, for a contrail to form, somewhere during the temperature and humidity decrease due to mixing, p^{H_2O} has to surpass p_s^{H2O} to become supersaturated. This requirement for contrail formation is generally accepted and is named the "Schmidt-Appleman Criterion" (SAC).

Whether the SAC is satisfied thus depends on the exact "route" of the exhaust from being hot and humid to being cold and dry. A convenient way to visualise the mixing and possible surpassing of p_s^{H2O} is a p^{H_2O} vs. T plot (see figure 2.1). Assuming an adiabatic and vapour conserving mixing process, during which heat and vapour diffuse at the same rate, the mixing can be represented by a straight line. This line, called the mixing line by Schumann (1996), connects the point representing the ambient conditions $\{p_\infty^{H_2O}, T_\infty\}$ and the point representing the conditions at the exit of the jet engine $\{p_j^{H_2O}, T_j\}$. The mixing line has slope G [Pa/K] (see equation 2.3).

$$G = \frac{p_j^{H_2O} - p_\infty^{H_2O}}{T_j - T_\infty} \tag{2.3}$$

Schumann (1996) found that the value of G can also be approached by a function of ambient conditions and combustion characteristics and formulated equation 2.4.

$$G = p_\infty c_p \frac{M(air)}{M(H_2O)} \frac{EI_{H_2O}}{(1-\eta)Q} \tag{2.4}$$

Where p_∞ [Pa] is the ambient pressure, c_p [kJ/kg K] is the air specific heat capacity, $M(air)$ [g/mol] is the molar mass of air and $M(H_2O)$ [g/mol] is the molar mass of water. Recall EI_{H_2O} as the water emission index, η as the overall propulsive efficiency and Q as the fuel's specific heat content.

The p^{H_2O} vs. T plot is composed of multiple important elements: firstly, the $p_s^{H2O}(T)$ line is plotted (purple) in figure 2.1). Secondly, the point of ambient p^{H2O} and temperature is depicted (grey dot with subscript ∞). Thirdly, the mixing line (grey) is drawn using the value of G. These three ingredients are sufficient to check if the SAC is satisfied and to conclude whether a contrail thus forms or not. For example, figure 2.1 shows a scenario (3) in which supersaturation w.r.t. water is not reached and a contrail does not form. Figure 2.1 also shows two scenario's (1 and 2) in which supersaturation w.r.t. water is reached (purple dots) and a contrail does form.

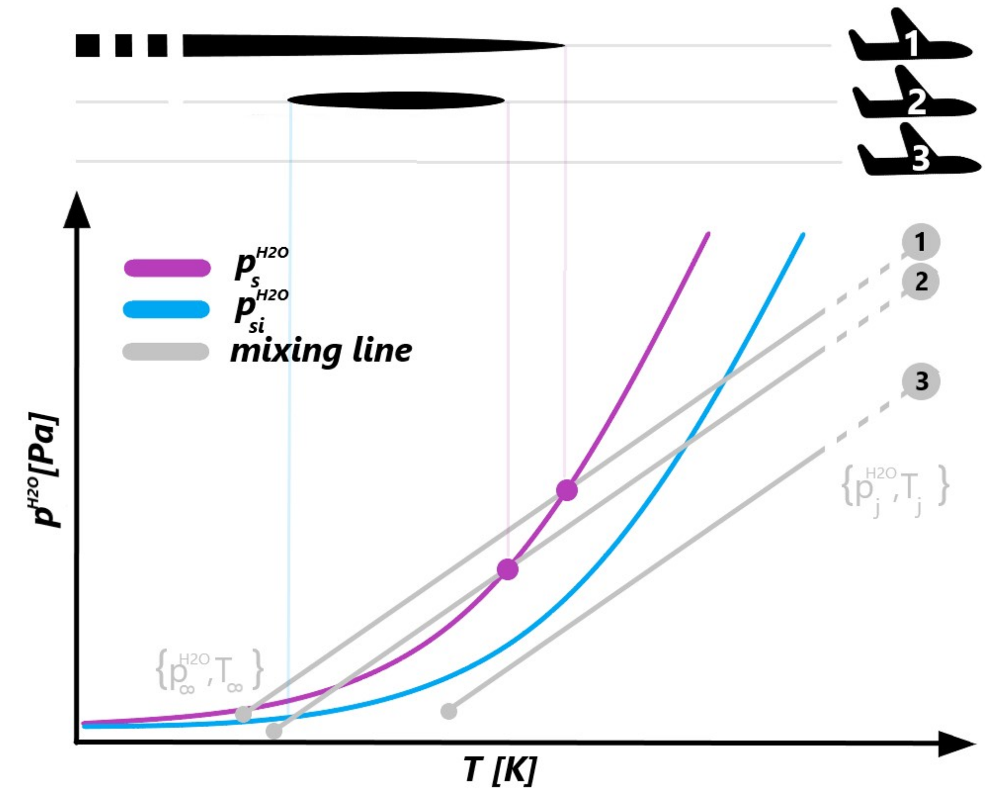

Figure 2.1: A p^{H_2O} vs. T diagram displaying three scenarios for the mixing of the aircraft exhaust with ambient air. Grey lines are mixing lines. Ambient and exhaust conditions are labelled with subscript ∞ and j, respectively. The purple line represents saturation with respect to water. The blue line represents saturation with respect to ice. A point of condensation is marked with a purple dot. Scenario 1 holds for an aircraft that forms a persistent contrail, scenario 2 holds for an aircraft creating a non-persistent contrail and scenario 3 shows a scenario in which no contrail forms. Additionally, whether a contrail forms and if it persists is represented by the illustration above the graph.

However, figure 2.1 features more elements that provide useful information. The first of those elements is the point representing engine exit conditions (grey dot with subscript j). As indicated by the dashed part of the mixing line, point $\{p_j^{H_2O}, T_j\}$ lies far outside the range of the pictured p^{H_2O} vs. T plot. The presence of point $\{p_j^{H_2O}, T_j\}$ is a helpful, visual reminder of the hot and humid origin of the exhaust gas. Typical exhaust conditions are, e.g., T = 800 K and p^{H_2O} = 1300 Pa. Lastly, the blue line, which represents the saturation with respect to ice, plays a crucial role in the persistence of a contrail. This will be discussed in the next section.

Concluding, this section identifies three ambient parameters found at the root of contrail climate impact, as they drive whether a contrail forms or not. Given a certain aircraft, the combination of ambient water vapour pressure, temperature, and air pressure determines whether or not a contrail can form.

2.1.2 Persistence

As discussed in the introduction this section, a contrail consists of frozen, nucleated droplets. A young contrail thus consists of ice-surrounded nuclei and can persist as long as these ice crystals remain. Whether the ice crystals of a contrail will last can be determined from the saturation with respect to ice (Brewer (1946), Schrader (1997)), represented by the blue line in figure 2.1. If the ambient air is supersaturated with respect to ice, the contrail's ice crystals will grow up to the point of equilibrium, that is, a relative humidity with respect to ice of 100%: in this case the young contrail will persist. The other scenario is that the ambient air is subsaturated with respect to ice, resulting in shrinkage of ice volume and finally its disappearance: this is called a non-persistent contrail. The top of figure 2.1 visualises the connection between contrail persistence, the mixing line and ambient conditions. Scenario 1 in figure 2.1 shows that a contrail can persist if the end of the mixing line is found above the ice saturation line. However, when the tail end of the mixing line is found below ice saturation, as seen for scenario 2, the contrail quickly dissolves. Naturally, the longer the lifetime of a contrail is, the larger is its climate impact.

This section shows a second dependence of contrail climate impact on ambient temperature and water vapour pressure. If the combination of these two parameters is found below ice saturation, the contrail ice crystals will shrink and the lifetime of the contrail will be short.

2.1.3 Contrail Spreading

After formation, a persistent contrail often does not maintain its recognisable linear shape. As a result of the vortex induced descend of the trail, the ice crystals of a contrail are exposed to horizontal wind. Horizontal winds usually have a velocity gradient along the vertical direction, which induces shear. This wind shear spreads the contrail to a width that can be 100 km (Kärcher (2018)). In jargon, the contrail (with the emphasis on "trail") loses its initial line shape and becomes a "contrail cirrus". In this way, the wind shear increases the width of the contrail cirrus. However, the chance exists that the wind drives the ice crystals outside contrail ice-supersaturated region, which leads to the sublimation of ice crystals and the disappearance of the contrail cirrus. An increase in contrail width increases the climate impact of the contrail. If the lifetime of a contrail is reduced because wind drives the contrail into ice-subsaturated regions, the contrail climate impact reduces as well.

Another dependence of contrail climate impact on an ambient factor is thus identified: wind shear and direction.

2.1.4 Optical Properties

Radiation passing through a contrail encounters a number of ice crystals. Upon this encounter, each of those ice crystals may prohibit radiation from continuing on its original path through scatter or absorption. This leads to the (partial) extinction of the radiation wave. To what extent a radiation wave extincts through the interaction with a contrail, depends on the optical properties of the contrail.

An intuitive factor that determines the extinction caused by a contrail is the density of ice crystals in a contrail. The higher the crystal density of a contrail, the larger the number of encountered ice crystals and the larger the extinction of the radiation will be. Unterstrasser and Gierens (2010) captured this relation in equation 2.5 for the extinction coefficient β_{ext} $[m^{-1}]$:

$$\beta_{ext} = IWC \times (u_1 + u_2/r_{eff}) \tag{2.5}$$

The equation features Ice Water Content (IWC), the ice crystal mass in kilograms per cubed meter, as a measure for contrail crystal density, u_1 and u_2 are constants, and r_{eff} is the effective radius of the ice particle distribution in meters. The latter parameter is used to represent the mixture of ice crystals with different sizes and shapes (habits).

The *IWC* of a contrail depends on a few factors, among which main ones are again temperature and p_{H_2O}. The combination of these two reoccurring parameters is the limiting factor for a contrail's *IWC* (Bock and Burkhardt (2016)). A larger availability of water vapour results in a larger average size and mass of ice crystals. Bier et al. (2017) found that the sedimentation rates of a contrail with heavier ice crystals is relatively large, resulting in an increase in *IWC* in its first hours ($< 4h$). Since a contrail with heavier ice crystals has a large falling rate, the growth in contrail volume is large, additionally ensuring the encounter of more fresh humid air for crystal growth. At a later stage the heavy ice crystals have the opposite effect. During the descent of the contrail, part of the ice crystals sublimate in ice-subsaturated air resulting in a decrease in *IWC*. Regarding r_{eff}, Bailey and Hallett (2004) found that the distribution of crystal sizes and habits largely depends on temperature and ice-supersaturation (and thus on water vapour pressure), but also on air pressure.

Ludlam (1980) found that the water vapour in the atmosphere available for condensation tends to rise with temperature. By Meerkötter et al. (1999), this was attributed to the increase in difference between the humidity limit for ice nucleation and the limit of ice saturation with temperature (Heymsfield et al. (1998)), which is an indicator for the water vapour available. Several field campaigns like the campaigns from Schiller et al. (2008), CRYSTAL-FACE (Gao et al. (2006)), SCOUT-O3 (Reus et al. (2009)) and CR-AVE (Flores et al. (2006)) provided in situ contrail data, allowing Schumann et al. (2017) to find a relation between *IWC* and *T*. This relation was captured by Schumann et al. (2017) in the following equation:

$$IWC = 10^{-8.7+0.04T} \tag{2.6}$$

Finally, β_{ext} can be multiplied with the geometrical thickness $D\,[m]$ of the contrail to arrive at the contrail optical depth, $\tau\,[-]$. One of the limiting factors for contrail thickness is the geometry of the ice-supersaturated layer in which the contrail is formed.

Another factor that affects the extent of radiation extinction is the angle of incoming radiation. This holds in particular for the contrail albedo, $\alpha_c\,[-]$, a parameter that represents the amount of radiation that is reflected by the contrail. For solar radiation, the contrail albedo depends on the cosine of the angle of the solar zenith ($\mu = \cos\theta$), the contrail optical depth, τ, and the ice particle effective radius, r_{eff} (Schumann et al. (2012)). The dependency of the albedo on μ introduces new dependencies on longitude, latitude, altitude and time of day. In light of the thesis goal, the dependency of contrail climate impact on μ will be separately and more extensively discussed in Section 2.1.7.

In short, this section shows another dependency of contrail properties on temperature and water vapour pressure. If the combination of temperature and water vapour pressure results in a large availability of water for condensation, the IWC and extinction coefficient of a contrail rises. The size and shape distribution of ice crystals are also dependent on temperature and water vapour pressure, and additionally on air pressure. Moreover, the geometry of the ice-supersaturated layer is limiting for contrail thickness. Lastly, dependency of optical properties on geographical location and time of day is also brought to the attention of the reader.

2.1.5 Long-Wave Radiative Forcing

Thus far, we have discussed the dependency of contrail RF on its formation, persistence, spread an optical properties. However, another important driver for the magnitude of contrail climate impact is the amount of radiation that would have passed through if the contrail were not present. Schumann et al. (2012) modelled the RF of contrails and included the undisturbed radiation in an insightful manner. Schumann et al. (2012) distinguished between the RF contribution for Long-Wave (LW) and Short-Wave (SW) radiation and formulated two separate equations accordingly. For clearness and conciseness the LW RF ($\mathrm{RF_{LW}}$) equation is only partly written below (equation 2.7). The equation for SW RF ($\mathrm{RF_{SW}}$) will be discussed in the next section.

$$\mathrm{RF_{LW}} = [\mathrm{OLR} - k_T\,(T_m - T_0)] \times ... \times E_{\mathrm{LW}}\,(\tau_c) \geq 0 \tag{2.7}$$

The first term of equation 2.7 accounts for the flux-change of outgoing LW radiation, assuming an opaque contrail. Appearing in this first term are the Outgoing Long-wave Radiation at the top of the atmosphere ($OLR\ [W/m^2]$), and T_m. T_m refers to the atmospheric temperature at the contrail mid layer. Equation 2.7 also features the parameters k_T and T_0. These two parameters are model parameters depending solely on the habits in question. Therefore, these parameters vary with temperature, water vapour pressure and air pressure, as was mentioned in Section 2.1.4. The larger the OLR, the larger the LW flux-change caused by a contrail can potentially be. However, the LW flux change is also dependent on T_m. If, due to its temperature, a contrail would emit the same amount of LW radiation into space as a clear sky, the contrail induced flux-change of outgoing LW radiation would be zero. Nonetheless, a contrail cirrus usually emits less LW radiation into space than a clear atmosphere. Through the Stefan–Boltzmann law this lower level of outgoing LW radiation can be related to the lower brightness temperature of contrails (Schumann et al. (2012)). Although they are not the same, the brightness temperature of a contrail is an indicator of its actual (mid layer) temperature and the other way around. Hence, the flux-change of outgoing LW radiation caused by the difference between the brightness temperature of a contrail and a clear atmosphere is approached with the term $k_T\,(T_m - T_0)$.

Then, through the last part of equation 2.7, Schumann et al. (2012) also accounted for the presence of other cirrus. $E_{\mathrm{LW}}\,(\tau_c)$ depends on the optical thickness of these other cirrus and makes sure that the LW contrail RF does not include RF caused by pre-existing cloud coverage. Therefore the contrail $\mathrm{RF_{LW}}$ also has a dependency on the temperature and p_{H_2O} at other levels of the atmosphere.

Concluding, ambient factors that affect the $\mathrm{RF_{LW}}$ of contrails are foremost OLR, temperature and pre-existing cloud coverage. Through the latter, additional affecting factors are temperature and water vapour pressure at other levels of the atmosphere. Lastly, parameters depending on the habits in the contrail account for the dependency on water vapour pressure and air pressure at the contrail level.

2.1.6 Short-Wave Radiative Forcing

Schumann et al. (2012) formulated equation 2.8 to calculate the $\mathrm{RF_{SW}}$ for a contrail. For this equation, α_c was already discussed in Section 2.1.4.

$$\mathrm{RF_{SW}} = -\,\mathrm{SDR}\,(t_A - A_{\mathrm{eff}})^2\,\alpha_c\,(\mu, \tau, r_{\mathrm{eff}})\,E_{\mathrm{SW}}\,(\mu, \tau_c) \tag{2.8}$$

Equation 2.8 starts with the dependency of $\mathrm{RF_{SW}}$ on the Sun Direct Radiation (SDR). The value for SDR is a multiplication of the solar constant, $S_0\ [W/m^2]$, and μ. The dependency of contrail climate impact on geographical location and time of day thus reoccurs here. Additionally, Schumann et al. (2012) accounts for the albedo of the Earth-atmosphere system (A_{eff}) that the SW radiation encounters if the contrail were not present. Since A_{eff} varies with surfaces type (snow, sea, forest, etc.), the dependency on longitude and latitude is emphasised another time. Equation 2.8 further shows dependencies of the $\mathrm{RF_{SW}}$ on the habit parameter t_A, the earlier discussed τ and again the presence of other cirrus (E_{SW}).

Thus, like $\mathrm{RF_{LW}}$ was dependent on the OLR, $\mathrm{RF_{SW}}$ depends on the incoming SDR. Habit dependent parameters re-emphasise the influence of temperature and water vapour pressure at the location of the contrail. The same two parameters at other levels of the atmosphere influence $\mathrm{RF_{SW}}$ through E_{SW}. Another new dependency on an ambient parameter is found in the albedo of the Earth-atmosphere system.

2.1.7 Day versus Night

We have seen that the RF_{SW} of a contrail, and thus also the total RF, is dependent on the presence and angle of daylight. Given the subject of this thesis, this section gives extra attention to the dependency of contrail RF on the time of day.

As described in Section 2.1.4, part of the dependence on daylight originates from the albedo of the habits in a contrail. This dependency is driven by the change in reflective properties of the ice crystals with θ. Another part of the daylight dependency simply originates from the magnitude of SDR, which is highest when θ equals 90 degrees and SDR is zero when the Sun is behind the horizon. Therefore (in combination with longitude, latitude and season) the time of day plays a major role in the net value of contrail RF. Schumann et al. (2012) presented results for equation 2.8 under realistic conditions (Schumann et al. (2012) figure 8) and clearly showed per habit how the RF_{SW} becomes increasingly negative with a more shallow angle of irradiance up to an angle of 75-80 degrees. Above an angle of 75-80 degrees the reduction in SDR starts to dominate the magnitude of RF_{SW} and consequently RF_{SW} approaches zero. Naturally, when the Sun is behind the horizon (i.e., nighttime), $\theta > 90^\circ$, which results in zero SDR and zero RF_{SW}. The results of Schumann et al. (2012) are consistent with findings by Meerkötter et al. (1999).

The negative RF_{SW} thus comes, varies and goes with the presence of the Sun. RF_{LW}, on the other hand, remains approximately constant throughout the 24 hours in a day (Schumann et al. (2012), Meerkötter et al. (1999)). In the presence of daylight, negative RF_{SW} can offset the positive RF_{LW} and even result in a net negative RF. The absence of negative RF_{SW} during nighttime leaves RF_{LW} uncompensated, inevitably resulting in a positive RF (Stuber and Forster (2007)).

The diurnal cycle of contrail cirrus RF is well captured in a figure presented by Meerkötter et al. (1999) (Meerkötter et al. (1999) figure 4a), which shows how the net RF varies with the time of day thanks to the variation of RF_{SW} whilst RF_{LW} remains constant. A close inspection of the figure of Meerkötter et al. (1999) even shows how the shape of the results of Schumann et al. (2012) reappears in the diurnal cycle. Logically, the net RF of a contrail depends on the degree of cancellation of RF_{LW} by RF_{SW} (Schumann and Graf (2013)). It is clear that the time of day has a significant impact on the contrail RF. This also implies the relevance of knowledge of contrail cirrus lifetime towards the quantification of contrail climate impact, as its RF may vary significantly throughout its existence, just as a result of the varying value of RF_{SW}.

2.2 Operational Measures

The spatiotemporal variability of contrail climate impact explained in the previous section has an important consequence: for a flight from airport A to airport B a trajectory exists that has a minimal contrail climate impact. By avoiding regions where warming contrails are formed and also possibly by deliberately flying trough regions where cooling contrails are formed, the overall temperature rise caused by contrails could be minimised. Although little of such efforts are made in practice until now, a growing body of research exist that has studied the opportunities of optimised trajectories. This chapter aims to present the opportunities for optimised trajectories in two steps: firstly, Section 2.2.1 discusses the application of altitude changes. Secondly, Section 2.2.2 will expand this to 3D trajectory optimisation. As the step from contrail formation to climate impact, or even RF, is a complex one, many of those studies limit themselves to the relation between operational mitigation measures and the potential contrail coverage. Other studies do present findings about the effect of these operational mitigation measures on the resulting climate impact.

2.2.1 Changing Altitude

Over the years, research confirmed that certain parts of the atmosphere feature a higher potential for persistent contrails than others. This potential is often expressed by the parameter Potential Contrail Coverage (PCC). The PCC can be defined as the maximum contrail coverage for an atmospheric state when aircraft fly through every part of it. Using the general circulation model ECHAM4 and an inventory of realistic flight movements, Fichter et al. (2005) studied the monthly mean values for PCC and observed that typical cruising altitudes largely overlap with altitudes with a relatively high PCC. Looking at the PCC from pole to pole, Fichter et al. (2005) concluded that a downward shift of trajectories could reduce the contrail coverage at tropical latitudes. For middle and high latitudes an upward shift of air traffic would result in less persistent contrails. Overall, the model of Fichter et al. (2005) predicted a 45% decrease in contrail coverage for a 6000 ft descent for all flight movements. However, Fichter et al. (2005) suggested that the findings described above do not hold throughout the whole year due to seasonal variability. E.g., in winter, a 6000 ft downward shift of high-latitude air traffic in the northern hemisphere results in increased contrail coverage. ATM aiming to reduce contrail coverage should thus account for the seasonal cycle, especially outside the tropical latitudes. According to Fichter et al. (2005) the 45% decrease in contrail coverage yields a reduction in contrail RF of comparable size, showing that an altitude change is an effective way to reduce contrail climate impact.

The findings of Fichter et al. (2005) on the change of contrail coverage with altitude were supported by earlier findings of Sausen et al. (1998). A recent study by Matthes et al. (2021) confirmed the changes in RF, reporting a global increase in RF of 3 mW/m^2 for a 2000 ft increase in altitude and a decrease of 5 mW/m^2 for a decrease in altitude of equal size. Avila et al. (2019) performed a study on the effect of an altitude increase on RF over the United States. Their results showed a 92% reduction in net RF when increasing the flight altitude by 4000 feet.

2.2.2 Changing 3D Trajectories

The vertical adjustment to the trajectory of a flight, as described in Section 2.2.1, is not the only change to the path of an aircraft that can prevent the formation of a contrail or mitigate the contrail RF. In addition, aircraft may combine the vertical manoeuvres with horizontal diversions to avoid ice-supersaturated regions. In other words, the trajectory of a flight can be changed in 3D to minimise its climate impact.

While the mitigating altitude changes of the previous section could partly rely on seasonal trends, this is a less suitable approach for 3D optimisation. The horizontal pattern of PCC or predicted contrail RF is rather spotty (see for example Yin et al. (2022)). In literature one thus only finds 3D mitigation attempts that make use of one day weather data, or an even higher time resolution, to model the actual airspace with its actual contrail supporting regions.

Multiple numerical studies have been performed to find the potential of 3D mitigation strategies. Some studies, like Yin et al. (2018) focused on the reduction of contrail coverage through 3D trajectory optimisation. Using air traffic optimiser and simulator AirTraf, in combination with EMAC Yin et al. (2018) found that a 90% decrease in contrail coverage could be achieved at the cost of a fuel penalty of only a few percent. These numbers are confirmed by several other studies like Campbell et al. (2008) and Yamashita et al. (2020).

Examples of studies that use the contrail climate impact, rather than PCC, as driver for trajectory optimisation are Zou et al. (2013) and Lim et al. (2017). Both studies confirmed that contrail climate impact can be reduced with limited extra costs. Other studies do not focus on contrail climate impact alone, but aim to reduce the total climate impact of a flight through trajectory optimisation. Such studies are that of Yamashita et al. (2020) and Yamashita et al. (2021), which showed by simulations with AirTraf that warming contrails were effectively avoided and cooling contrails effectively employed to reduce the overall climate impact of a flight. Using the Trajectory Optimisation Module (TOM) Lührs et al. (2020), Matthes et al. (2020) and Lührs et al. (2021) carried out optimisations for a selected winter day, trading-off total climate impact against fuel use. The results showed that an increase in fuel use of 0.75% could already lead to a 50% decrease in total climate impact, dominated by the mitigation of contrail climate impact. These results confirmed again that the reduction of contrail climate impact is an efficient way of mitigating the total impact of aviation.

Chapter 3

Methodology

This chapter describes the methodology applied to achieve the goal of this thesis. A high-level flow chart of the research method is provided in figure 3.1. The flow chart starts with two green blocks at the top of figure 3.1, which represent the preparatory steps for the thesis research. The blue blocks show actions that produce the research data and steps that process this data to enable meaningful analysis. Lastly, the three orange blocks represent the data analysis phase and each block leads to the fulfilment of one of the thesis objectives as formulated in Section 1.2. To provide the reader with relevant technical background, this chapter starts with description of the applied modelling chain in Section 3.1. This modelling chain is the main source of data towards completing the research objectives. The rest of the chapter continues along the outline of figure 3.1. Section 3.2 covers the preparatory steps of the thesis that led to adequate simulation settings. Section 3.3.2 describes how the difference between daytime and nighttime in contrail ATR20 for eco-efficient trajectories will be analysed. Next, Section 3.3.3 describes the analysis of the variability of the contrail ATR20 of eco-efficient trajectories with a changing threshold time until sunrise for nighttime contrails. Lastly, Section 3.3.4 presents the approach to explore a climate optimal way to spend a given SOC budget to reduce contrail ATR20.

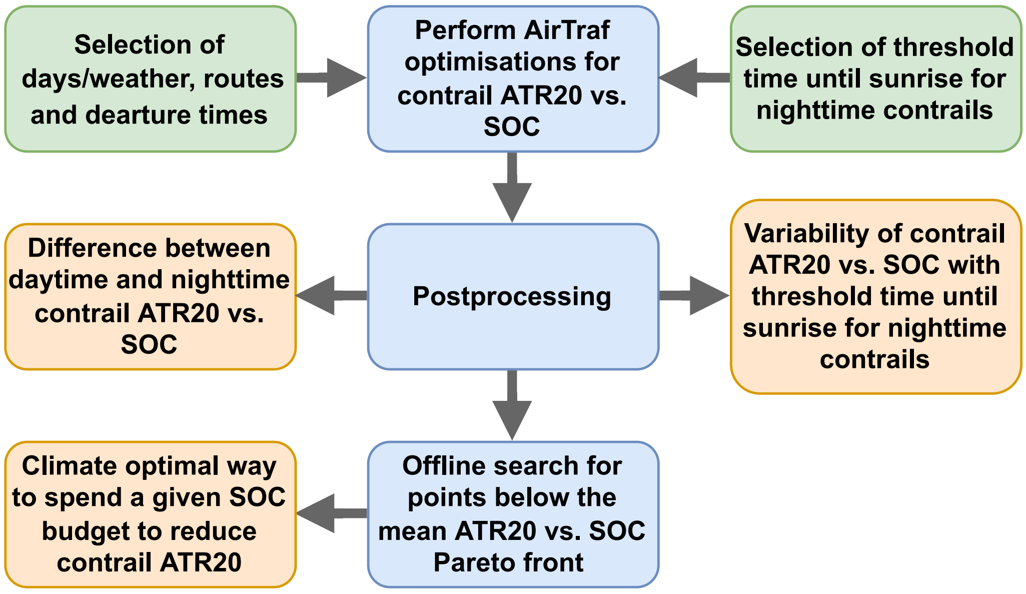
Selection of days/weather, routes and dearture times
Perform AirTraf optimisations for contrail ATR20 vs. SOC
Selection of threshold time until sunrise for nighttime contrails
Difference between daytime and nighttime contrail ATR20 vs. SOC
Postprocessing
Variability of contrail ATR20 vs. SOC with threshold time until sunrise for nighttime contrails
Climate optimal way to spend a given SOC budget to reduce contrail ATR20
Offline search for points below the mean ATR20 vs. SOC Pareto front

Figure 3.1: Flowchart of the thesis methodology.

3.1 Modelling Chain

While tests to climate-optimise the trajectory of real-world flights are slowly taking off[1], current experiments in this field are mostly based on numerical simulations and optimisations. Making use of the numerical approach comes at a cost: findings in a simulated world are not guaranteed to hold in the real world. However, numerical simulations also come with a set of benefits: time, airspace and departure slots are scarce in real-life air traffic, but are virtually limitless for a numerical simulation. This means that we can simulate a flight at the time, location, and conditions that we prefer, being only limited by the available computational resources. This also means that experiments are easily repeated under the exact same conditions, making it easier to analyse the effects of an operational or technical measure. In the ever changing atmosphere of the real world, it is impossible to exactly copy an experiment. An additional plus of a numerical simulation is that it is performed without any emissions or climate effects in the real world. Hence, FlyATM4E and this thesis make use of numerical tools to answer their research questions.

A tool that is suitable for the numerical optimisation of flight trajectories is EMAC coupled with the submodels AirTraf, CONTRAIL and ACCF. Depending on the selected AirTraf routing option, this model combination searches for trajectories with, for example, minimal climate impact or minimal economic cost and simulates the flights correspondingly. This section aims to present the working principles of the AirTraf submodel and the connected (sub)models (Yamashita et al. (2023)). To this purpose, the section starts with a description of the base model EMAC in Section 3.1.1. In Section 3.1.2 we describe the derivation of the aCCFs, and in particular of the contrail aCCF. The section continues with an overview of the CONTRAIL submodel in Section 3.1.3. Lastly, Section 3.1.4 describes how the AirTraf model is set up and how the model performs a trajectory optimisation.

3.1.1 Base Model

The AirTraf model is a submodel of the ECHAM/MESSy Atmospheric Chemistry (EMAC) model. EMAC is able to numerically simulate climate, chemistry and transport, covering the troposphere and middle atmosphere including their interaction with land, oceans and human influences (Jöckel et al. (2010)). As can be deduced from its acronym, EMAC is composed of two main elements. The 5th generation of the European Centre Hamburg circulation model (ECHAM5) makes up the core atmospheric model of EMAC (Roeckner et al. (2003), Roeckner et al. (2006)). The ECHAM5 version applied in this thesis will be 5.3.02. The second element is the Modular Earth Submodel System (MESSy), of which version 2.54.0, will be applied. This part of the EMAC model is responsible for the coupling of submodels to ECHAM5 (Jöckel et al. (2010)). MESSy provides access to several dozens of multi-institutional submodels.

A user of EMAC has a choice between several spatial resolutions. The resolution for this thesis study is chosen as T42L31ECMWF, which is in line with earlier studies of the (Fly)ATM4E project (e.g., Castino et al. (2022), Yamashita et al. (2020)). T42L31ECMWF means that horizontally the grid is constructed by a spherical truncation of T42, which amounts to a Gaussian grid of approximately $2.8^\circ \times 2.8^\circ$ latitude and longitude (or 300 km X 300 km). L31ECMWF refers to the 31 vertical hybrid pressure levels of the grid. The top of the grid is found at 10 hPa, corresponding to a total height of approximately 30 km and levels of approximately 1 km. Weather data for the simulations is imported from the European Centre for Medium-Range Weather Forecasts (ECMWF) ERA-Interim reanalysis data, towards which the EMAC model dynamics are nudged down to the surface.

For this thesis, EMAC is thus used to simulate Atmospheric conditions which are forced to be close to the actual weather patterns, enabling a realistic experiment that takes these

[1]`https://satavia.com/news/`, accessed: November 29, 2022.

conditions into account. Furthermore, the EMAC model allows for repeated simulations under the same weather conditions, making it possible to observe the effect of changes to flight strategies or to the simulation set up.

3.1.2 Algorithmic Climate Change Function

As stated in the research goal and thesis title, this thesis deals with optimised trajectories. To optimise aircraft trajectories for the purpose of climate impact mitigation, first, a tool is needed to quantify the climate change resulting from aircraft emissions and climate effects. This tool should account for position, time and conditions (e.g., the conditions provided by EMAC). Furthermore, the tool preferably performs the climate impact quantification at acceptable computational cost. This tool is provided in the form of aCCFs by the ACCF submodel. This section is used to explain the origin and derivation of these functions. In light of the thesis, extra attention will be given to the aCCF associated with contrail climate impact.

The aCCFs find their origin in Climate Change Functions (CCFs) derived within the European REACT4C (Reducing Emissions from Aviation by Changing Trajectories for the benefit of Climate, Matthes (2012)) project. These functions are a collection of 4D (longitude, latitude, altitude and time) data sets and describe the climate change induced by a single local flown kilometer or by a local unit of emission (Grewe et al. (2014b)). The data set of the CCFs was constructed using a collection of eight representative weather situations for the North Atlantic region (Grewe et al. (2017), Frömming et al. (2020)). While simulating those weather situations in EMAC, emissions were released at several hundred time-region grid points (168 grid points at 3 different emission times, Grewe et al. (2014b)). Contrail formation and the impact on atmospheric concentrations due to the emitted species was tracked using a Lagrangian scheme. Finally, the resulting RF could be obtained.

The found values for RF could be fed into a multitude of emission scenario's and climate metrics to come to a measure of the resulting climate change. For example: a metric could be used that assumes a pulse emission and expresses the Absolute Global Warming Potential (AGWP) at 100 years from the emission. Alternatively, the chosen metric could describe the Average Temperature Response (ATR) over a 50 year period, resulting from a maintained volume of air traffic rather than a pulse emissions. Which metric is the right choice depends on the question one wants to answer with the metric. For this purpose Grewe et al. (2014b) formulated a political question to identify the right metric for REACT4C: "What is the short-term climate impact of consistently applying the REACT4C re-routing strategy?". The question led to F-ATR20 as the metric of choice for the CCFs of REACT4C. The metric uses a 20 year time horizon, which is a fitting horizon to indicate short-term climate impact and is especially suitable for contrail related research, as contrails in particular have a short lifetime compared to other climate effects. In addition, the F-ATR is an eligible choice to capture the climate effect of actual air traffic and its future (F) volume, compared to a scenario with a sustained level of air traffic or a pulse emission (Grewe et al. (2014b)).

For a given RF the ATR20 is obtained by first calculating the temperature response, ΔT [K], (see equation 3.1).

$$\Delta T = \lambda \cdot RF \tag{3.1}$$

Where λ $[K/(Wm^{-2})]$ is the climate sensitivity parameter. Secondly, equation 3.2 is applied, resulting in the desired ATR20.

$$ATR20 = \frac{1}{20}\int_{t_0}^{t_0+20} dT(t)dt \tag{3.2}$$

This value per kilogram emission $[K/kg]$ or per flown kilometer $[K/km]$ is finally the CCF. To test the sensitivity of the results to the choice of climate metric, other suitable metrics (like pulse AGWP20) where tested to evaluate the impact of a change in routing strategy, but did not significantly affect conclusions (Grewe et al. (2014a)).

A drawback of the CCFs is that they can only be applied in the specific regions and weather patterns in which they where conceived, and obtaining them is a computationally expensive process (van Manen and Grewe (2019)). Towards real-time trajectory optimisations, a tool is required that can asses the climate impact in a less time consuming manner. This tool was found in the form of the aCCFs. The philosophy behind the aCCFs is to discover the link between meteorological key parameters and the climate impact of emissions in order to enable the quick calculation of the CCF for real-time meteorological data (Matthes et al. (2017)). The aCCF for CO_2 was found with the nonlinear climate–chemistry response model AirClim (Yamashita et al. (2020)). ACCFs for ozone, methane and water vapour were developed by van Manen (2017) and van Manen and Grewe (2019) via a regression analysis of the CCF data. Yin et al. (2022) presents the complete collection of aCCFs and its supplement of discusses the derivation of the contrail aCCF in detail. This derivation is revisited below.

The approach to find the aCCF for contrails was based on the method of van Manen (2017) to derive the aCCFs for water vapour, methane and ozone. The procedure can be summarised as the construction of several scatter plots of the contrail net RF versus a set of key meteorological parameters. To find the parameters to which the net RF has the strongest relationship, a multitude of fits to the data was found through regression methods. The supplement of Yin et al. (2022) first describes how a data set containing contrail net RF was constructed. Similar to other aCCFs, the starting point of the contrail aCCF derivation are Lagrangian trajectories. These trajectories were simulated by Irvine et al. (2014) for a set of winter days above the Atlantic ocean. To determine the characteristics of persistent contrails Yin et al. (2022) made use of several parameters provided by Irvine et al. (2014). Among others, these parameters included location, time of day, contrail lifetime and initial temperature. These parameters allow the estimation of SDR. OLR values were taken from the ECMWF Integrated Forecast System.

To arrive at an estimate for contrail RF, Yin et al. (2022) made use of the equations of Schumann et al. (2012) as presented in Section 2.1.5 and Section 2.1.6. The set of parameters provided by Irvine et al. (2014) does not suffice to fill out all parameter required by the equations of Schumann. Hence, Yin et al. (2022) had to assume an ice particle effective radius (r_{eff}), which leads to an estimate of ice habit mixture using Schumann et al. (2011) and Schumann et al. (2017). Further necessary assumptions are that of contrail depth, contrail width and the absence of overlapping cloud coverage. The lifetime of contrails was assumed to be 6 hours.

Now a data set of contrail RF values could be computed with the equations of Schumann et al. (2012). Given that daytime contrails interact with SW radiation in addition to LW radiation, the search for a proper fitting to the RF data set was split up in a fitting for daytime contrails and a fitting for nighttime contrails. Every contrail that was formed during daytime, even if it had a part of its 6 hour lifetime at night, was marked as a daytime contrail. Any contrail with its complete lifetime during night was referred to as a nighttime contrail. Contrails that were formed during night but persisted into the day were not regarded.

For nighttime contrails, temperature was identified as the parameter with the highest correlation to contrail net RF. Correspondingly, a suitable exponential relation with temperature was found for the nighttime aCCF. The exponential fit ruled out the possibility of negative RF values and additionally follows logically from the way temperature enters the RF parameterization of Schumann et al. (2012). For daytime contrails, OLR was shown to have the highest rank correlation coefficient. The daytime aCCF followed from a linear regression of the RF onto OLR. The equations for daytime and nighttime hold for the RF of a 6 hour contrail, but should represent the ATR20 to agree with the other aCCFs. This is achieved by a simple scale factor of 0.0151 [$K/W/m^2$]. The contrail aCCF, both for day and night, is presented in equation 3.3 (Yin et al. (2022)). The resulting value of equation 3.3 is the contrail aCCF in $K/km(contrail)$. The symbol T represents the atmospheric temperature in Kelvin and OLR stands for Outgoing Long-wave Radiation in W/m^2. The nighttime aCCF is not valid for temperatures below 201

K and should be equalled to zero in the rare case that a colder temperature is encountered. Moreover, the function makes sure that when a point in flight features no Potential persistent contrail cirrus coverage (Potcov), aCCF$_{\text{contrail}}$ is zero.

$$\text{aCCF} = \begin{cases} 10^{-10} \times \left(0.0073 \times \left(10^{0.0107\times T} - 1.03\right)\right) \times 0.0151, & \text{if Potcov} > 0 \text{ and nighttime,} \\ 10^{-10} \times (-1.7 - 0.0088 \times \text{OLR}) \times 0.0151, & \text{if Potcov } > 0 \text{ and daytime,} \\ 0, & \text{if Potcov } \leq 0 \end{cases} \tag{3.3}$$

Equation 3.3 provides the contrail ATR20 caused by a pulse emission (P-ATR20). However, to evaluate the long term effect of a policy or technical change, one is often interested in an increasing emission scenario that considers the growth of air traffic. This is also the case for this thesis. Hence, a metric conversion factor of 13.6 is applied (Yin et al. (2022)). Moreover, the efficacy of contrail RF with respect to other forcing agents is not accounted for in equation 3.3. Therefore, the outcome of equation 3.3 is additionally multiplied with the contrail efficacy factor of 0.42 presented by Lee et al. (2021).

3.1.3 CONTRAIL

From equation 3.3 it becomes apparent that the correct prediction of contrail aCCF requires knowledge of the Potential Contrail Coverage (PCC) of the atmosphere. Within the system of EMAC and its submodels, CONTRAIL (supplement of Grewe et al. (2014b)) is the submodel that is responsible for the calculation of the potential coverage of persistent contrails. This section aims to give the reader insight in how this submodel performs the PCC calculation.

For the evaluation of PCC it is necessary to know whether a created contrail adds to the total cloud coverage or that its effects are negligible due to preexisting cloud coverage. Since cloud and contrail formation is a process at subgrid scale, a parameterization for a proper grid scale approximation of contrail contribution to cloud coverage is required. Burkhardt et al. (2008) provided such a parameterization based on relative humidity, starting with the following relation for natural cloud coverage, b_{ci}:

$$b_{ci} = 1 - \sqrt{1 - \frac{r - r_{ci}}{r_{sat} - r_{ci}}} \tag{3.4}$$

When r, the EMAC grid mean relative humidity, is larger than r_{ci}, the critical relative humidity for cloud formation, cloud coverage assumes values larger than zero. If the cloud coverage is indeed larger than zero, its exact value is driven by the ratio $(r - r_{ci})/(r_{sat} - r_{ci})$, where $(r_{sat} - r_{ci})$ represents the difference between the relative humidity at saturation (r_{sat} = 1) and again the critical relative humidity. Full cloud coverage is reached when r equals r_{sat}. Each relative humidity in this section refers to that over ice, as temperatures at typical cruising altitudes ($<$ 241 K) only allow for a small number of supercooled water droplets (Burkhardt et al. (2008)).

Next, the maximum possible coverage of both natural clouds and persistent contrail cirrus, b_{co+ci} is computed by equation 3.5 as given by Burkhardt et al. (2008).

$$b_{co+ci} = \begin{cases} \dfrac{r - r_{co}}{r_{sat} - r_{ci}} - b_{ci} \cdot (1 - b_{ci}) & \text{for } r_{co} \leq r \leq r^{*} \\ 1 & \text{for } r > r^{*} \end{cases} \tag{3.5}$$

Here r_{co} represents the critical relative humidity. If the mean relative humidity of an EMAC grid box is above r_{co}, part of the grid box is ice-supersaturated. $r^{*} = r_{sat} - (r_{ci} - r_{co})^2 / (r_{sat} - r_{ci})$ is a threshold value for the relative humidity above which the maximum possible coverage of both natural clouds and contrail cirrus equals 1. r_{co} is computed via equation 3.6.

$$\frac{r_{co}}{r_{ci}} = \frac{r_{SAC}}{a \cdot r_{nuc}} \tag{3.6}$$

Where r_{SAC} is the relative humidity at which contrails form during the mixing of jet and ambient air, r_{nuc} is the homogeneous freezing threshold and a can be regarded as a tuner for a realistic contrail parameterization. Persistent natural and contrail cirrus can form at lower values for relative humidity than the homogeneous freezing threshold, hence a assumes values smaller than 1.

Now, the PCC can be calculated by subtracting the natural cloud coverage from the maximum possible coverage of natural clouds and contrail cirrus combined. Following the described parameterisation, the PCC thus provides the fraction of the EMAC grid box that can potentially be covered by contrail cirrus (see equation 3.7).

$$PCC = b_{co+ci} - b_{ci} \tag{3.7}$$

Burkhardt et al. (2008) found a realistic PCC for a equalling 0.9 by comparing the results of the parameterisation with earlier in situ measurements and satellite estimates (e.g., Gierens et al. (1999)). The CONTRAIL submodel finds the PCC for the EMAC grid boxes and passes this to the AirTraf submodel. The next steps are carried out by the AirTraf submodel itself.

The approach of the CONTRAIL submodel has been verified by Frömming et al. (2011) using the Benchmark test of Myhre et al. (2009). A reasonable agreement was shown with other models and measurements. CONTRAIL showed a higher frequency of contrails with low *IWC* (and lower optical depth accordingly) than observational data. Yin et al. (2018) stated that this can be either attributed to inadequacy of observational techniques that fail to detect contrails with low optical depth, or that the difference can originate from a model bias. Furthermore, Yin et al. (2018) simulated the PCC for a full year using CONTRAIL and observed that the submodel is well able to capture the seasonal cycles of PCC as described in earlier work (e.g., Fichter et al. (2005)).

3.1.4 AirTraf

AirTraf 3.0 (Yamashita et al. (2023)) is an EMAC submodel for 4D simulation and optimisation of air traffic trajectories, including the effects of actual and local weather conditions in its calculations. AirTraf 3.0 is currently under development and features new modules for Optimisation and Decision-Making problems that are more efficient that AirTraf 2.0. AirTraf 3.0 will be available in the next MESSy release, but some of the new modules are already used for this thesis work. This section aims to present the working principles behind AirTraf and how it interacts with the base model EMAC and submodels CONTRAIL and ACCF. Again, given the context of the thesis, extra attention is given to contrail related procedures.

AirTraf follows the general work flow depicted in figure 3.2, adapted from Yamashita et al. (2015). At the initialisation of AirTraf the air traffic data is imported. This data contains the airports of departure and arrival, their respective location and the departure time of each flight. Additionally, information about the aircraft and its performance and the user's choice of routing option is read in. The block "Decomposition of trajectories" in figure 3.2 refers to the parallelisation of the trajectories by the message passing interface. At every time step in EMAC's time loop, AirTraf checks for each flight if its time of departure is reached. When the departure time is indeed reached, AirTraf calculates the trajectory conform the chosen routing option, for example an optimisation w.r.t. Simple Operating Costs (SOC), contrail ATR20 or a trade-off between objectives. Once the optimal trajectory is computed, the aircraft follows this trajectory, moving along its line with each time step. The fuel use of the aircraft is found through the use of a total energy model, which finds its basis in aircraft performance data provided by the Base of Aircraft Data (BADA) of Eurocontrol (EUROCONTROL (2011)). Engine emission data by the ICAO in combination with the DLR fuel flow method (Deidewig et al. (1996)) are used to find the NO_x emissions along the trajectory. This procedure is performed for every flight. Finally, an output file is created with the relevant information such as costs, emissions, flight time, contrail distance and climate impact. AirTraf restricts itself to the cruise phase and hence does not optimise nor compute the take-off and approach phase of a flight.

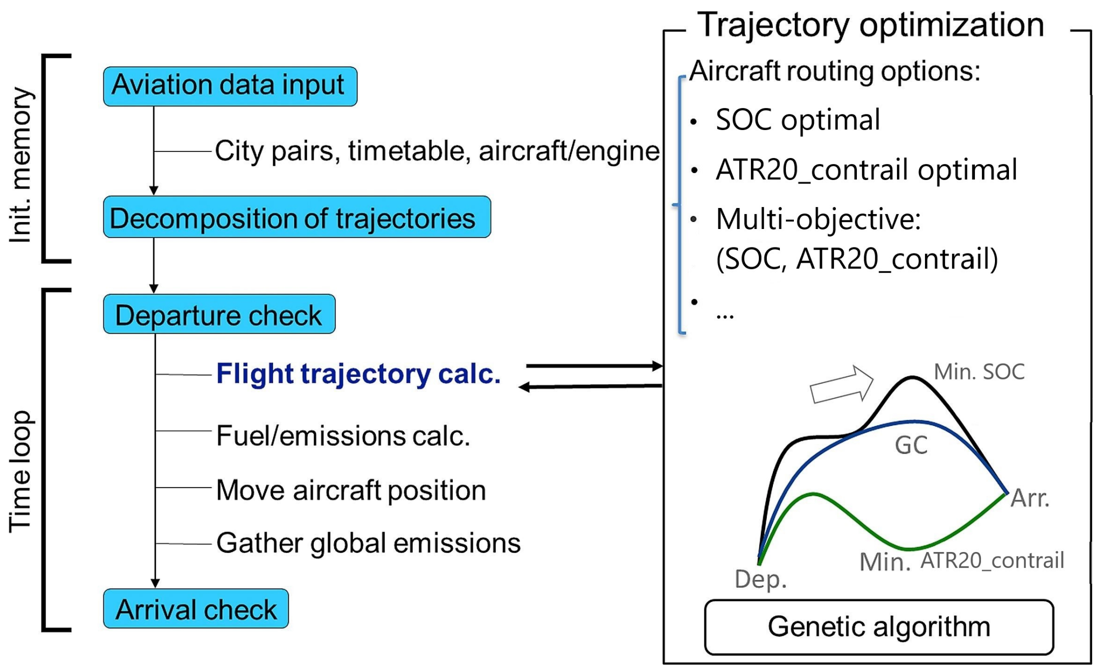

Figure 3.2: Overview of the AirTraf simulation tool (adapted from Yamashita et al. (2015)).

To understand how AirTraf optimises and calculates its trajectory, it is important to understand the trajectory parameterization used by AirTraf. The reference trajectory between a given city pair is given by the coordinates of that city pair and by the Great Circle (GC) line that connects them. The optimisation algorithm then makes use of eight Control Points (CPs) to reshape the trajectory and find the optimal trajectory (see figure 3.3). Three of the eight CPs are 2D coordinates (longitude and latitude) that control the horizontal trajectory of the flight, while the other five CPs are 1D controllers of the vertical trajectory. This means that AirTraf deals with 11 independent design variables $X = (x_1, x_2, ...x_{11})^T$ to shape the flight trajectory. The CPs are restricted in their movement by a domain. The domains of the horizontal CPs have a width of $0.1 \times \Delta\lambda_{airport}$ in longitude and a width of $0.3 \times \Delta\lambda_{airport}$ in latitude, where $\Delta\lambda_{airport}$ is the longitudinal distance between the airport pair of the flight. The centres of the three domains are fixed at three nodes that divide the GC trajectory in four parts of equal length. The five vertical CPs are longitudinally fixed at the five nodes that would divide $\Delta\lambda_{airport}$ in six parts equal in longitudinal distance. They are restricted to flight levels between FL290 and FL410. Both horizontally and vertically, the trajectory is represented by a third-order B-spline curve with as controllers the three horizontal CPs and five vertical CPs, respectively. Naturally, the start and end of the trajectory are fixed at the airport pair of the route in question.

The optimisation of a trajectory is done with the help of the Adaptive Range Multi-objective Genetic Algorithm (ARMOGA) version 1.2.0 by Sasaki and Obayashi (2005). Starting from an initial random population, the ARMOGA iterates the trajectory simulation. Through evaluation, selection, crossover and mutations multiple generations of trajectories are created. Both the size of the initial population and the number of generations are predefined. Together they determine the size of the total pool of solutions. A proposed trajectory consists of 101

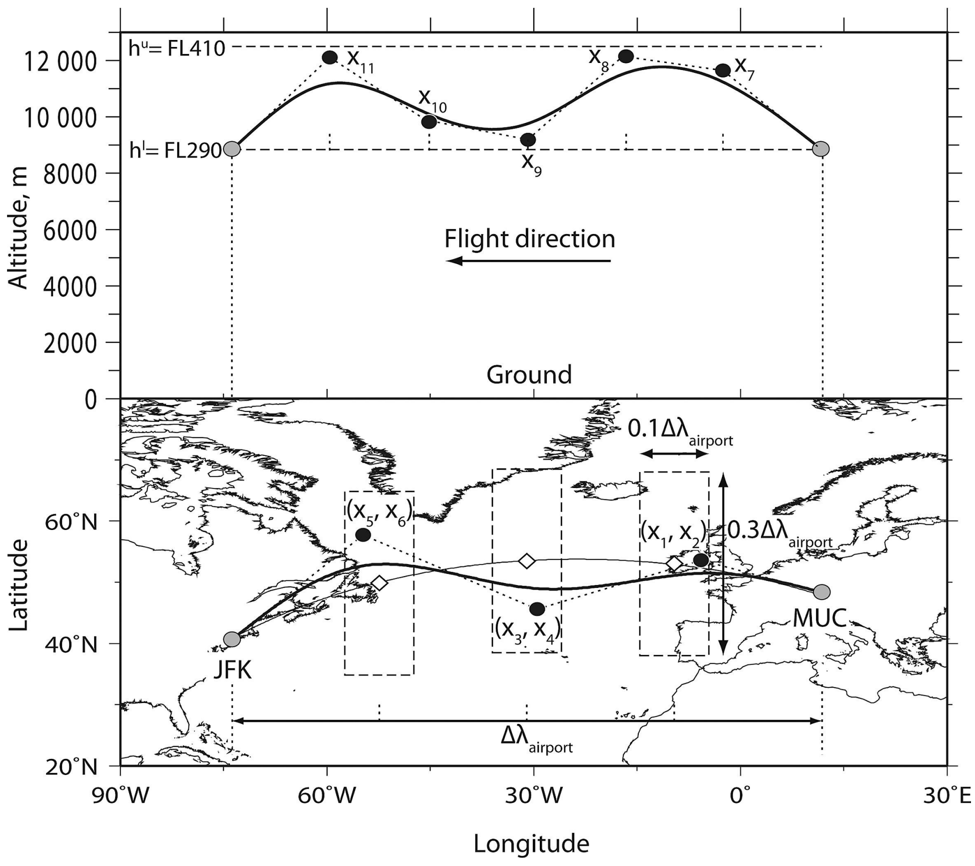

Figure 3.3: Example geometry definition for a flight trajectory in the vertical cross-section (top) and the projection on Earth (bottom). The bold solid line indicates the optimised trajectory. The control points are represented by black dots. The dashed lines and boxes show the domains of the control points. The diamonds along the great circle are the center points of the horizontal domain boxes (Yamashita et al. (2016)).

waypoints and the 100 segments in between. The waypoints and segments of the trajectory are used for the evaluation of the objective function in service of the optimisation process. For the first performance evaluation of the different solutions, AirTraf keeps the weather conditions fixed as they are at the time of departure. Considering the optimisation objective, the ARMOGA identifies the solution with the best value and stores the corresponding trajectory (Yamashita et al. (2020)). Upon the actual execution of the flight, the previously fixed weather conditions are disregarded. For the evaluation of, for example, the fuel use or ozone climate impact, the real-time weather conditions are thus applied, which change per EMAC time step.

In case the selected routing option includes contrail distance or contrail climate impact, the output of the CONTRAIL submodel becomes relevant (Section 3.1.3). Having the PCC for an EMAC grid box quantified with the CONTRAIL submodel, the next step is to translate this to the contrail coverage of a proposed trajectory. This is described by Yin et al. (2018) and is briefly visited below. As depicted in figure 3.4a, the value for PCC of the EMAC grid box closest to the waypoint is transferred to said waypoint. Next, all waypoints are associated with a flight distance. For waypoint i, this flight distance is the sum of half the length of segment $i-1$ (the segment before waypoint *i*) and half the length of segment *i* (the segment after waypoint *i*). This is shown in figure 3.4b. Note that for the first waypoint only half the length of the following waypoint is associated, whereas for the last waypoint only half the length of the previous section is counted. The contrail distance for the i^{th} waypoint, $PCCDist_i$, is thus

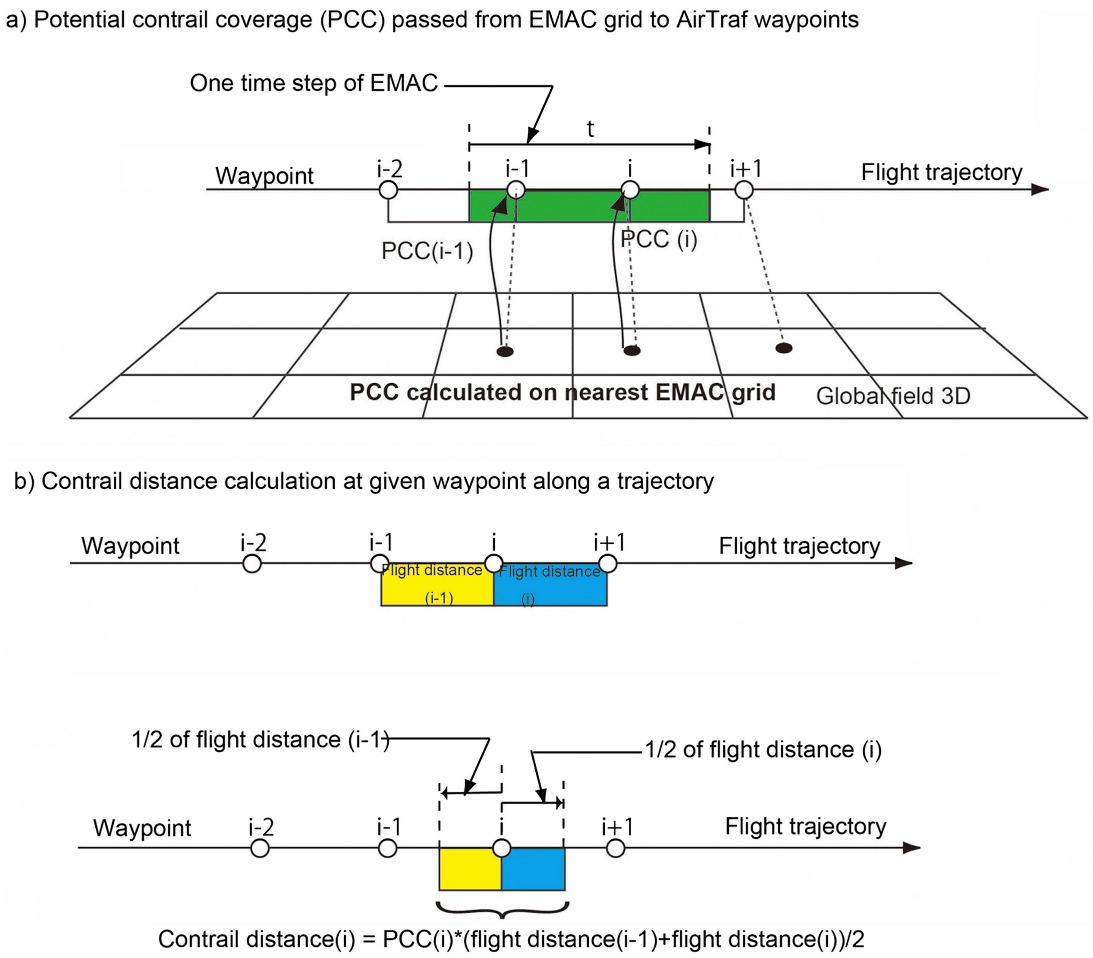
a) Potential contrail coverage (PCC) passed from EMAC grid to AirTraf waypoints
One time step of EMAC
t
Waypoint
i-2
i-1
i
i+1
Flight trajectory
PCC(i-1)
PCC (i)
PCC calculated on nearest EMAC grid
Global field 3D
b) Contrail distance calculation at given waypoint along a trajectory
Waypoint
i-2
i-1
i
i+1
Flight trajectory
Flight distance (i-1)
Flight distance (i)
1/2 of flight distance (i-1)
1/2 of flight distance (i)
Waypoint
i-2
i-1
i
i+1
Flight trajectory
Contrail distance(i) = PCC(i)*(flight distance(i-1)+flight distance(i))/2

Figure 3.4: Illustration of the contrail distance calculation procedure. (a) PCC exported from EMAC grid to waypoints along the trajectories. (b) Contrail distance calculation at a waypoint (i) (Yin et al. (2018)).

calculated as:

$$PCCDist_i = PCC_i \cdot \frac{FlightDist_{i-1} + FlightDist_i}{2} \tag{3.8}$$

Then the total contrail distance of a trajectory, $PCCDist_{tot}$, is found by summing the contrail distance of each of the 101 waypoints:

$$PCCDist_{tot} = \sum_{n=1}^{101} PCCDist_{(n)} \tag{3.9}$$

Furthermore, AirTraf computes for every segment the associated values of aCCF, among which we also find the contrail aCCF (see Section 3.1.2). The contrail ATR20 per segment is simply computed as a product of the contrail aCCF at the i^{th} waypoint and the PCC_{dist} at the i^{th} segment (note the subtle difference with $PCCDist_i$, which is associated with a waypoint) (Yamashita et al. (2020)).

$$ATR20_{contrail,i} = aCCF_{contrail,i} \times PCC_{dist,i} \tag{3.10}$$

A chosen routing option in AirTraf often comprises a trade-off between an indicator of climate impact, like the ATR20 of a species or $PCCDist_{tot}$, and a kind of cost. This cost can be an increase in climate impact of another effect, but more often the mitigation of climate impact is set off against an increase in time, fuel use, cost or a combination of those (e.g., Yin et al. (2018), Castino et al. (2022)). Within this thesis, the cost is estimated in terms of Simple Operational Cost, or SOC. The SOC is therefore shortly explained below.

SOC is a simple definition of the cost to operate an aircraft, taking into account both time and fuel consumption through a linear relationship. To express the contribution of both factors in a total cost in dollars, the time and fuel used to travel the i^{th} segment ($TIME_i$ and $FUEL_i$) are summed for the total trajectory and then multiplied by their respective unit costs, $c_t = 0.75\$/s$ and $c_f = 0.51\$/kg$. These cost coefficients, used by AirTraf, were taken from Burris (2015). The corresponding is equation is shown in equation 3.11.

$$SOC = c_t \sum_{i=1}^{100} TIME_i + c_f \sum_{i=1}^{100} FUEL_i \tag{3.11}$$

Table 3.1 provides an overview of the outputs of AirTraf that are relevant for this thesis. The table is divided in three categories. At the top the reader finds those properties that are collected from the nearest EMAC grid box at the time of flight departure, the second group of rows consists of properties calculated along with the trajectory, and the bottom rows are properties that are found with the so-called "fuel–emissions–cost–climate calculation module" of AirTraf. This last category contains properties like fuel use and SOC. Depending on the selected routing option, some of the properties in the last category are already calculated in flight trajectory optimisation step. There are three types of attributes: behind those attributes that are associated with a waypoint the reader will find a "W", those associated with a flight segment are marked with and "S", and the attributes that belong to complete trajectory are followed by the letter "T".

Table 3.1: Relevant properties assigned to a resulting flight trajectory in AirTraf 2.0. The properties of the three groups (top to bottom, divided by horizontal lines) are, respectively, obtained from the nearest grid box of EMAC, the flight trajectory calculation, and the fuel–emissions–cost–climate calculation. The attribute type indicates where the values of properties are allocated. "W", "S", and "T" in column 3 stand for waypoints, flight segments, and a whole flight trajectory, respectively. (Adapted from Yamashita et al. (2020))

Property	Unit	Attribute type	Description
$aCCF_{contrail}$	$K(km(contrail))^{-1}$	W	aCCF of contrails[a]
$Potcov$	fraction	W	Potential persistent contrail cirrus coverage[b]
$ATR20_{contrail}$	K	S	Anticipated climate impact of contrails
$ATR20_{total}$	K	S	Anticipated climate impact (total)
d	m	S	Flight distance
ETO	Julian date	W	Estimated time over
h	m	W	Flight altitude
PCC_{Dist}	km(contrail)	S	Contrail distance[c]
λ	deg	W	Longitude
ϕ	deg	W	Latitude
F_{cr}	$kg(fuel)s^{-1}$	W	Fuel flow of an aircraft (cruise)
$FUEL$	kg	S	Fuel use
SOC	USD	T	Simple operational cost

[a]Yin et al. (2022); [b]Frömming (2014);[c]Yin et al. (2018).

Yamashita et al. (2020) and Yamashita et al. (2016) described how the results of AirTraf compare with literature like that of Sridhar et al. (2013) and Rosenow et al. (2017): although the study states that direct comparison is difficult due to difference in methodologies, relative changes predicted by AirTraf are largely in agreement with literature.

3.2 Simulation Set-Up

While Section 3.1 described the combination of EMAC with the submodels AirTraf, CONTRAIL and ACCF, this section will describe how this tool will be applied to produce data to address our research goal. Since computational and storage resources are not unlimited, choices have to be made to limit the amount of simulations and the amount of generated data to a manageable amount. Additionally, simulation and optimisation settings should be chosen such that the simulation results in meaningful data. This section describes and motivates the chosen set up of the simulations that are carried out for this thesis. Section 3.2.1 describes which days are chosen for the simulation. Section 3.2.2 provides the applied flight plan. Section 3.2.3 discusses the selection of a set of values for the threshold time until sunrise for nighttime contrails. Lastly, Section 3.2.4 motivates the choice in AirTraf optimisation settings.

3.2.1 Day Selection

To properly account for the day to day weather variability and the seasonal cycle, flights are simulated over a multitude of days. Ideally, the collection of days contains all 365 day for a multitude of years. However, resources limit the number of days to analyse. Hence, a representative selection of days is required for this thesis. The starting point for the day selection of the simulations is the data from an AirTraf 2.0 simulation from December 1st 2017 until December 1st 2018. This simulation was carried out using an European flight plan comprising the top 100 European routes in terms of available seat kilometers in 2018. All the flights depart at 12.00 and are simulated with an A320 aircraft model. Two optimisations were carried out for this one year simulation: one with a minimum total ATR20 as the objective (climate optimal), and one with minimum SOC as the objective (SOC optimal).

As a first step, the results of the climate optimal and the SOC optimal simulation are compared by computing the total contrail ATR20 per day, d. Next, the difference in daily contrail ATR20 between the two optimisations is found with equation 3.12. The result is plotted and shown by the grey line in figure 3.5.

$$\Delta ATR20_{contrail,d} = (ATR20_{contrail,d})_{clim.opt.} - (ATR20_{contrail,d})_{cost.opt.} \tag{3.12}$$

Although the climate optimal solution involves the mitigation of the ATR20 induced by all climate forcers, this optimisation was very much driven by the mitigation of contrail ATR20 (as also seen in Yamashita et al. (2021)). Hence, the calculated values for $\Delta ATR20_{contrail,d}$ $[K]$ are indicative for the mitigation potential for contrail ATR20 individually. Days that show a large negative $\Delta ATR20_{contrail}$ have a large potential to mitigate the climate impact of contrails, while mitigation efforts are little rewarding on days where $\Delta ATR20_{contrail}$ has a small magnitude.

In the selection of simulation days, the main goal is to arrive at a selection that allows for the most general conclusions in the analysis phase of this thesis. Towards this selection, we note that the potential of contrail formation is characterised by a seasonal cycle. Since the thesis focuses on European air space, we are dealing with a domain of mid-latitudes in the northern hemisphere. For this domain the PCC is largest in winter and lowest in summer (Yin et al. (2018), Fichter et al. (2005)). The PCC for spring an autumn is found in between those extremes. Conform this finding, candidate days for the selections were only sought in winter (December, January, February) and summer (June, July, August). Just as for PCC, it is expected that the results of this thesis for spring and autumn should be in between the results for winter and summer. Hence, winter and summer simulations allow for a derivation of conclusions for the two remaining seasons. Moreover, the aCCFs are derived for summer and winter, which makes their use more reliable for these two seasons than for spring and autumn.

Taking into account computational resources, it was decided to select approximately one

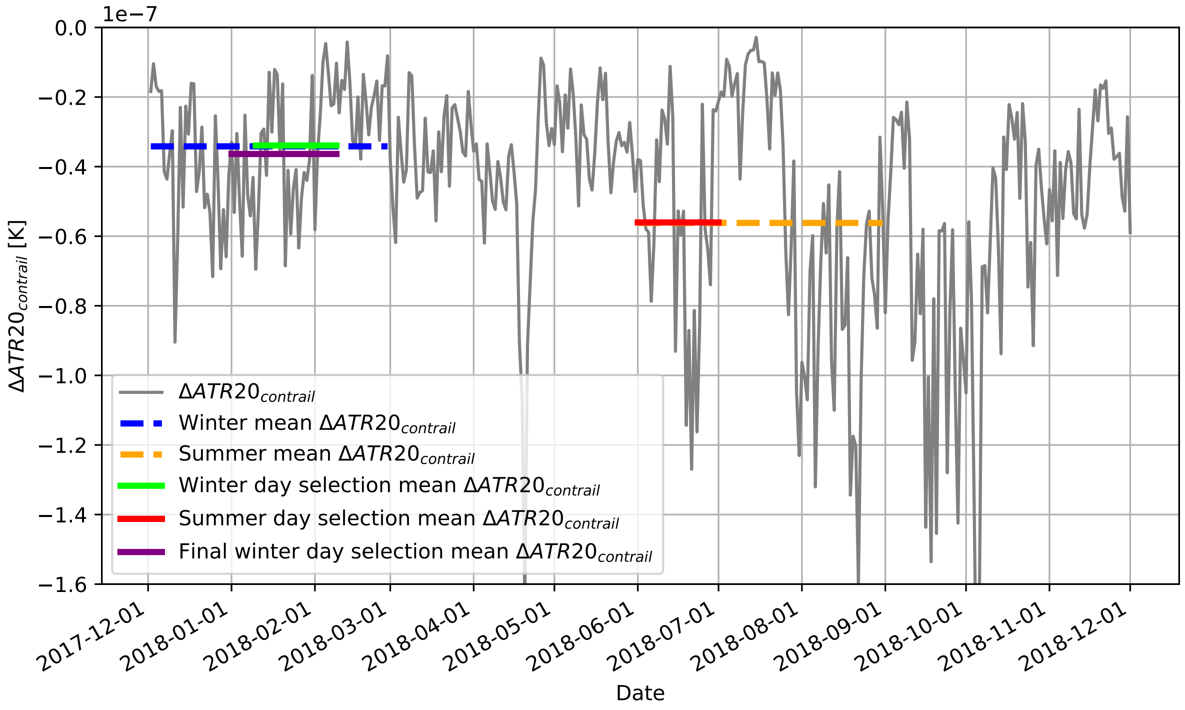

Figure 3.5: Daily values for $\Delta ATR20_{contrail}$ leading to the selected days for the thesis simulations.

month of consecutive days for both winter and summer. To find the most representative day selection, the mean value of $\Delta ATR20_{contrail}$ was used as an indicator. For every possible selection of 31 consecutive days in winter and summer, the mean $\Delta ATR20_{contrail}$ was compared to the mean $\Delta ATR20_{contrail}$ of the whole respective season. The mean values of $\Delta ATR20_{contrail}$ for winter and summer are shown in figure 3.5 by a blue and an orange dotted line, respectively. The 31-day selection with the mean $\Delta ATR20_{contrail}$ closest to the seasonal mean was chosen as most representative for the season. For winter, this resulted in a 31 day range from 2018-01-10 until and including 2018-02-09 (see green line in figure 3.5). For summer the period from 2018-06-01 until and including 2018-07-01 was selected (see red line in figure 3.5). For efficiency reasons in the preparation of the simulation, the start of the winter simulations was shifted to 2018-01-01, resulting in a 40 day winter period (see purple line in figure 3.5).

Whether the variance of $\Delta ATR20_{contrail}$ of the final day selections matched the variance featured by the corresponding season was checked. The variances were judged to match reasonably well. The mean and variance of $\Delta ATR20_{contrail}$ for winter, summer and the final day selections are given in table 3.2. Here, variance is defined as the average of the squared deviations from the mean.

Table 3.2: $\Delta ATR20_{contrail}$ mean and variance for winter, summer and the final day selections.

Parameter	**Winter**	**Winter selection (relative difference)**	**Summer**	**Summer selection (relative difference)**
Mean $\Delta ATR20_{contrail}$ $[K]$	$-3.42 \cdot 10^{-8}$	$-3.64 \cdot 10^{-8}$ (-6.4%)	$-5.62 \cdot 10^{-8}$	$-5.61 \cdot 10^{-8}$ (+0.3%)
Variance $\Delta ATR20_{contrail}$ $[K^2]$	$3.36 \cdot 10^{-16}$	$3.18 \cdot 10^{-16}$ (-5.3%)	$13.6 \cdot 10^{-16}$	$9.04 \cdot 10^{-16}$ (-33.9%)

In table 3.2 can be seen that the mean $\Delta ATR20_{contrail}$ of the summer selection matches better with the seasonal mean than the mean of the winter selection. The winter selection has a mean that is lower than the seasonal mean, while the summer selection shows a mean that is slightly higher than the seasonal mean. For the later presented results, this means for winter that the observed magnitude of contrail ATR20 mitigation is possibly overestimated by a few percent. For summer the opposite holds: the results could underestimate mitigation potential, although the summer defect is much smaller than that of winter. For the parameter "variance $\Delta ATR20_{contrail}$" the winter selection corresponds better to its respective season than the summer selection. The variances of both selections are lower than the seasonal variances. For the results of both seasons this means that the uncertainty of the mitigation potential is larger than one could calculate based on the produced data alone. This especially holds for summer, and to a lesser extent for winter.

3.2.2 One Day Flight Plan

The selection of flights for the research of this thesis is driven by an aim for both a geographical and temporal spread. Hence, valuing both aims equally, four routes were selected which are all flown four times a day. The four departure slots are fixed at UTC midnight, at 6.00 UTC, at 12.00 UTC and at 18.00 UTC, achieving an even spread of flights over the day. The spread out departure times ensure that different types of flights will occur: flights completely during daytime, flights completely during nighttime, flights starting in daytime but ending in nighttime and of course, flights starting in nighttime but ending in daytime. This is key towards the achievement of the thesis goal, which aims to find the difference between daytime and nighttime.

The four selected routes are chosen from the ATM4E flight plan and are shown in figure 3.6. The selection shows a good coverage of the European continent. Regarding the mitigation of contrail ATR20, previous analysis within our research group suggested that three regions with distinct trends can be identified within the ATM4E set of routes (F. Castino personal communication). The selected routes cover two of those regions well: Western Europe (route 1) and Central Europe (routes 2, 3 and 4). Routes in the third region, South-East Europe, are missing from the route selection. North-south-wise good coverage is important, as longitudinally the duration of daytime and nighttime varies considerably. With a latitude of 63°, Trondheim is even close to latitudes where the phenomena of polar night (in winter) and midnight Sun (in summer) start to occur.

Moreover, the route selection shows diversity in orientation. Routes 1 and 2 fly a longitudinal trajectory, whilst routes 3 and 4 are oriented laterally. To also have a variation in flight direction, route 1 is heading from the north to the south, while route 2 is planned the other way around. Similarly, route 3 is directed from east to west, while route 4 will be going from west to east. Considering the sunrise and sunset, we thus have one route (route 3) meeting the terminator (the line between day and night) head on and one route (route 4) flying away from it. The orientation of routes 1 and 2 causes these routes to be more or less passively captured by the terminator, rather than approaching or outrunning it.

An attentive reader may have noticed that the number of routes is considerably smaller than the number of simulated days. This is motivated by the accent of this study, which mostly focuses on the dependency of the mitigation potential on the background weather pattern, rather than on the route. The simulation of flights is carried out on 71 days and for four departures per day, resulting in a total of 284 instances of the four routes. Hence, using the proposed days in combination with the one day flight plan, the number of temporal evaluations is large compared to other studies like Lührs et al. (2021) or Yin et al. (2018). Finally, the total number of flights is 4 x 284 = 1136 flights.

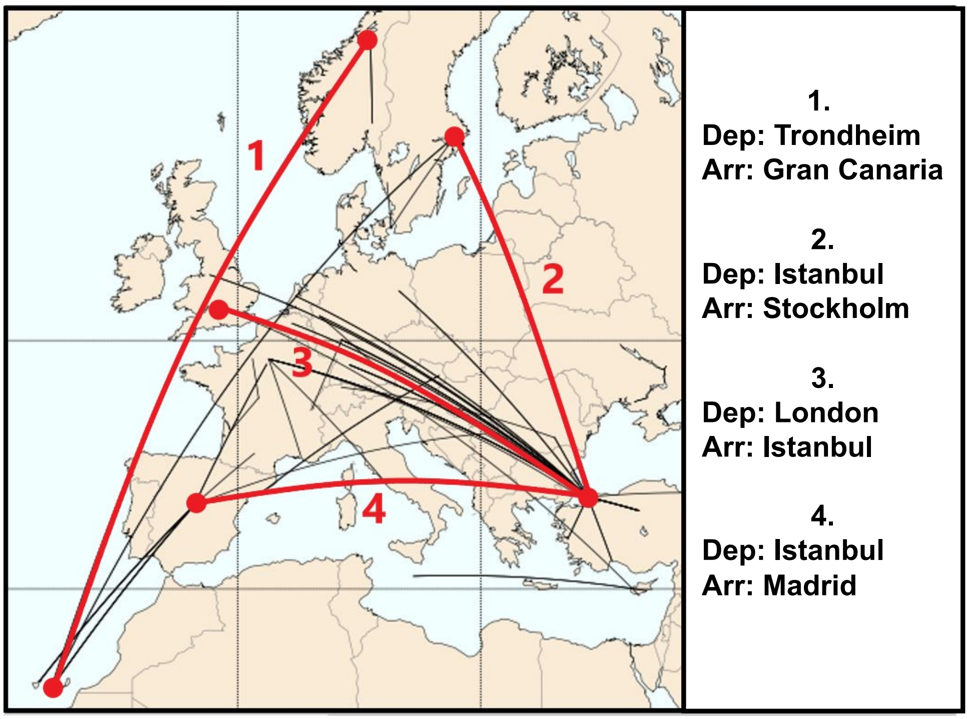

Figure 3.6: Selected routes for the flight plan (adapted from FlyATM4E internal communication).

3.2.3 Threshold Time Until Sunrise for Nighttime Contrails

The element that finally determines the total number of simulations is the Threshold time until Sunrise ($TH_{sunrise}$). As formulated in the second thesis objective, $TH_{sunrise}$ will be varied to analyse the sensitivity of the results to this parameter. This section will first describe the role of the $TH_{sunrise}$ within AirTraf and then explain how this parameter will be varied for the sensitivity study.

When AirTraf evaluates the contrail ATR20 of a flown flight segment, the submodel checks the parameters that are relevant for the aCCF calculation. As described in equation 3.3, these parameters include T, OLR, Potcov and the time of day. Whether the time of day is characterised as "daytime" or "nighttime" depends on the time at the moment of formation and the time until sunrise. If it is daytime, the daytime contrail aCCF is applied. The same holds if it is nighttime and the time until sunrise is smaller than the $TH_{sunrise}$. If it is nighttime, and the time until sunrise is greater than the $TH_{sunrise}$, the nighttime contrail aCCF is applied. ACCF's procedure for the distinction between daytime and nighttime is summarised by figure 3.7.

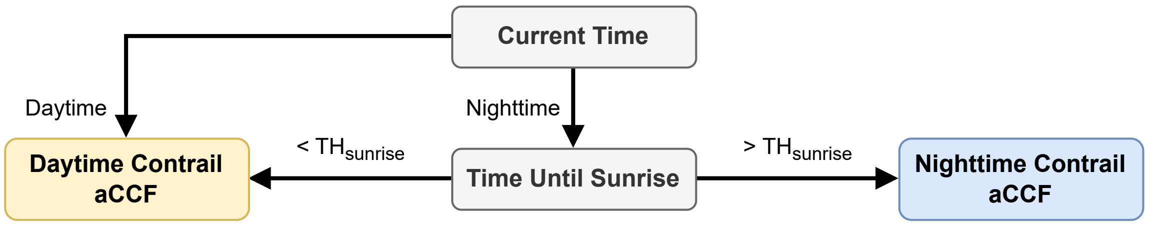

Figure 3.7: Visualisation of the application of threshold time until sunrise ($TH_{sunrise}$) in ACCF.

The simulations for the thesis will be performed for three different $TH_{sunrise}$ values: 0 hours, 3 hours and 6 hours. Simulations with $TH_{sunrise}$ = 0h will be considered as the baseline simulations for this thesis. A $TH_{sunrise}$ of 0 hours has as a consequence that the time of formation directly determines whether the contrail is treated as nighttime or daytime contrail. The choice for a $TH_{sunrise}$ of 3 and 6 hours is motivated by the way the contrail aCCF was derived. In the development of the aCCF Yin et al. (2022) assumed a contrail lifetime of 6 hours. A $TH_{sunrise}$ of 3 hours thus means that when the largest part of a contrail's 6 hour lifetime is in daytime, the daytime contrail aCCF is applied correspondingly. Similarly, when the largest part of a contrail's lifetime is in nighttime, the nighttime contrail aCCF is applied. Lastly, applying the 6 hour $TH_{sunrise}$ means that if only a small fraction of a contrail's total lifetime is in daytime, the contrail will be treated as a daytime contrail.

Varying $TH_{sunrise}$ tests the sensitivity of the results to this parameter directly, but the sensitivity to assumed contrail lifetime is tested indirectly as well. Given $TH_{sunrise}$ = 0h and that the formation of a contrail happens a minute before sunrise, figure 3.7 prescribes that the contrail is treated as a nighttime contrail. If the actual lifetime of a contrail is 12 hours instead of the currently assumed 6, the mark 'nighttime contrail' becomes very peculiar (if it was not already): only a minute of the contrail's 12 hour lifetime is spend during nighttime. A higher $TH_{sunrise}$, e.g., $TH_{sunrise}$ = 6h, could correct for the peculiar 'nighttime contrail' categorisation, since the 12 hour contrail would only be marked as 'nighttime contrail' if more than half of its life is found before sunrise. Regarding the distinction between daytime and nighttime of ACCF, a large $TH_{sunrise}$ can be seen as more correct for contrails with a large lifetime, while a small $TH_{sunrise}$ is more appropriate for short-living contrails. Thus, although a direct variation of the assumed contrail lifetime should involve a change in the contrail aCCF (equation 3.3), varying only $TH_{sunrise}$ already sheds light on the answer to the question: what if the assumed contrail lifetime is varied?

With the final selection of different values for $TH_{sunrise}$, the total number of flights to be simulated and optimised by AirTraf comes down to (31+40)x4x4x3 = 3408 flights.

3.2.4 Optimisation Settings

Now that the days, routes, departure times and the set of $TH_{sunrise}$ values are selected for simulation and optimisation by AirTraf, the exact settings for optimisation need to be selected. The routing option selected for the thesis is a trade-off between contrail ATR20 and SOC. The selection of the contrail ATR20 as an optimisation driver ensures that the effect of only contrail climate impact on optimised trajectories is captured. Other options would make it more difficult to interpret the results coming forth of the simulations. E.g., the selection of the total ATR20 routing option instead, would make it hard to unambiguously attribute observed trends and effects to just the contrail climate impact. Selecting the routing option for minimum total PCC distance would fail to take the contrail climate impact into account during the simulations. Moreover, the key difference between daytime and nighttime contrail climate impact would not be reflected in the optimisation process. By using the SOC as a second driver of the optimisation, both the change in time and fuel are weighed in the analysis. The combination of the two objectives will result in insight in the tension between the minimisation of costs and the minimisation of climate impact.

3.3 Data Analysis

Now that the set-up of the simulations is established in the previous sections, the simulations can be carried out. To make the raw data outputted by AirTraf more suitable for conclusion drawing, the data needs to go trough an analysis approach. The approach consists of multiple approaches and steps that are described in this section. To this end, Section 3.3.1 will first describe the raw data that is provided by the AirTraf simulations. This section is followed by an explanation of the data analysis used to achieve research objectives one, two and three in Section 3.3.2, 3.3.3 and 3.3.4, respectively.

3.3.1 Raw Data

For this thesis, AirTraf is tasked to perform an optimisation with respect to two objectives: minimal contrail ATR20 and minimal SOC. Fulfilling this task, AirTraf provides its user with solutions for each flight that are Pareto optimal. That is: those solutions that cannot lower the contrail ATR20 of a flight without increasing its SOC, neither vice versa. For most flights, AirTraf finds multiple Pareto optimal solutions that together form a Pareto front. Some solutions for a flight will feature relatively high SOC paired with relatively low contrail ATR20, other solutions will show a mediocre performance for both aspects, while other solutions will show relatively low SOC but a contrail ATR20 that is relatively high. Often, more than a hundred Pareto optimal points are found by AirTraf, but in some cases only one solution is found. For these flights, reaching a minimal SOC does not conflict with the objective to minimise contrail ATR20. In practice this means that the flight is contrail-free and that no reasonable flight diversion can result in a cooling contrail.

For each of the solutions of a flight, AirTraf outputs the following information: the coordinates of each of the 8 CPs that shaped the trajectories, the total value for SOC and the total contrail ATR20. We call this collection of data “flight data”, as they provide data on the level of a flight in its entirety. Moreover, AirTraf allows the user to select a set of solutions about which AirTraf outputs more detailed flight properties, namely per waypoint. We call this data “waypoint data”. For this thesis, the waypoint data is requested from AirTraf for three solutions: the SOC optimal solution, the contrail ATR20 optimal solution (in short: contrail optimal solution) and a solution in between that corresponds to approximately a 0.5% increase in SOC w.r.t. the SOC optimal solution. For flights with only two solutions, the middle of the three solutions will coincide with the closest of the other two points. Flights with only one solution will naturally have all three solutions coinciding. An example sketch of a set of Pareto efficient solutions is provided in figure 3.8. The example shows for which solutions AirTraf provides flight data (blue), and for which solutions both flight data and waypoint data is provided (blue-orange).

The two types of data partly allow for similar kinds of analysis. Flight and waypoint data are both suitable for the reconstruction of the trajectory that belongs to a solution and can both provide the total SOC and contrail ATR20 of a flight. However, there are also differences between the two data types. For most flights, the number of solutions stored in the flight data is well above 10, thus a Pareto front can be plotted with a high sample frequency. This means that we can get a clear picture of how the SOC increases with the mitigation of contrail ATR20. This correlation is defined at a lower resolution for the three-point waypoint data. Although the flight data prevails over the waypoint data in terms of sample frequency along the Pareto front, it is the the latter data that has the upper hand in terms of data quantity. Foremost, the data per waypoint allows the analysis of distinct parts of a trajectory. This can, for example, be used to see which parts of a flight are in nighttime or in daytime, which is useful towards the goal of this thesis. Secondly, besides coordinates, contrail ATR20 and SOC, the waypoint data also provides information about the ATR20 of other climate forcers, contrail distance, fuel use and more. This extra data expands opportunities for analysis, although this will be little used in this work. Corresponding with the strengths and weaknesses of the two data types, flight and waypoint data will be employed differently in the thesis analysis. How they are employed and why, will be explained per analysis in the upcoming sections.

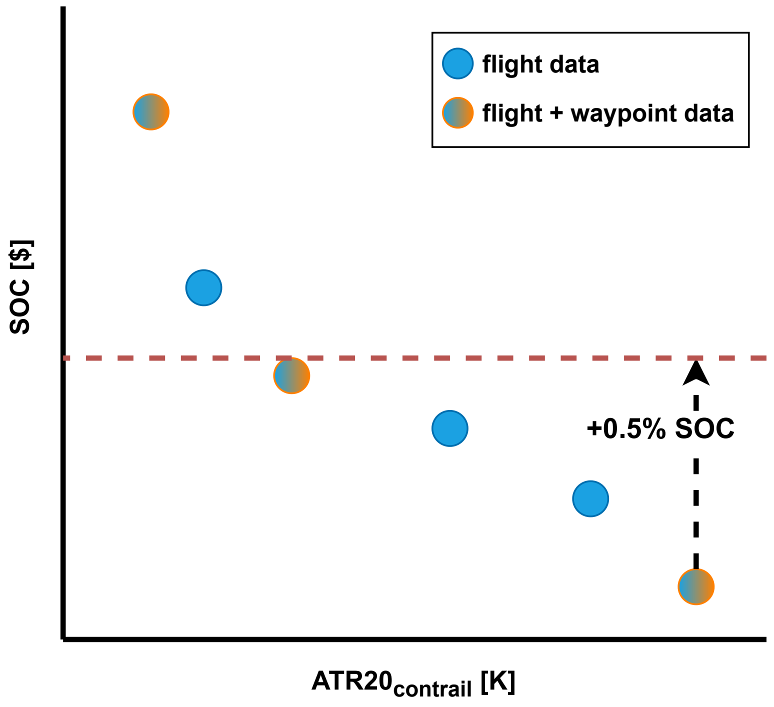

Figure 3.8: Sketch of a possible set of Pareto efficient solutions for a flight. A distinction is shown for solutions that are included in the flight data (blue), and solutions that are included in both the flight and waypoint data (blue-orange).

3.3.2 Data Analysis: Daytime and Nighttime Contrail Climate Impact

The first objective of this thesis is to analyse the daytime and nighttime difference in contrail ATR20 for eco-efficient trajectories. To analyse the difference between daytime and nighttime contrail climate impact, five different analyses will be used. Each of them should highlight different aspects of the day-night difference. For this purpose, both flight data and waypoint data will be processed. The way data is prepared for day-night analysis will be explained below.

Cost Optimal Specific Contrail ATR20 - Waypoint Data

The first approach that will be used to analyse the difference between daytime and nighttime contrail climate impact, is a day-night comparison of the magnitude of contrail ATR20. The analysis will only be performed for the SOC optimal trajectories. Since AirTraf does not take the resulting contrail ATR20 into account while shaping the trajectories, one can regard the associated values for contrail ATR20 as samples that represent the European contrail aCCF. The found values should indicate the original day and night contrail ATR20 of European flights, which is the starting point for mitigation. Values for contrail ATR20 shall be obtained from the waypoint output of AirTraf. We will pay no attention yet to the possible mitigation of contrail ATR20 with increasing SOC. In other words, this analysis has no interest in the shape of the Pareto front.

As explained in Section 3.3.1, waypoint data allow for the separation of daytime segments from nighttime segments within a trajectory. The waypoint data do not directly provide information on whether the ACCF submodel has treated a flight segment as a daytime or a nighttime segment. Hence, this information has to be retraced, which is possible by using the time stamp, longitude, latitude and altitude of the waypoints. To find the time of sunset and sunrise for a given day and location, the python suntimes 1.1.2 module is applied[2]. Four parameters are calculated for each flight segment as the average of its two adjacent waypoints: the local time, the time of sunrise, the time of sunset and the time until the first next sunrise. These parameters suffice to walk through the decision diagram presented in figure 3.7, and thus each segment can be classified as daytime or nighttime segment accordingly. The described approach is not guaranteed to exactly copy the day-night division made by ACCF. To minimise the number of wrong classifications during analysis, an extra condition is applied: if a segment features negative contrail ATR20, it is classified as a daytime segment, since negative ATR20 cannot occur during nighttime.

Having the the segments classified, we could now sum the nighttime contrail ATR20 and the daytime contrail ATR20 and compare the two. However, it is very possible that the amount of kilometers flown during nighttime does not equal the amount of kilometers flown during daytime. In winter, nights are relatively long, possibly resulting in more kilometers flown during nighttime. This causes a bias towards nighttime contrail ATR20. Similarly, a bias towards daytime contrail ATR20 could occur for summer. To eliminate the chance that a higher total ATR20 is merely the consequence of a larger amount of kilometers, we divide the sum of contrail ATR20 for daytime and nighttime by the total number of flown daytime and nighttime kilometers, respectively. We finally arrive at the specific contrail ATR20, for which the symbol $\overline{ATR20}_{contrail}$ $[K/km]$ will be used.

The results shall be plotted as a day and a night bar plot. The results will be shown for winter and summer data separately. This should allow for distinction between winter and summer behaviour for the optimisation studied in this thesis.

Average Pareto Fronts - Waypoint Data

The second analysis approach will pay attention to the day-night difference when it comes to the mitigation of contrail ATR20. For this purpose, not only the cost optimal solution is inspected, but also those solutions that are more contrail optimal. Therefore, this analysis involves Pareto fronts, aiming to give insight in the mitigation potential for flying in daytime or nighttime and the cost-efficiency of this mitigation. This analysis can be performed with either waypoint data or flight data. Since both kinds of data allow for different insights, analysis will be carried out for both, starting with the waypoint data.

Similar to the previous analysis, this analysis is also prone to a bias caused by a difference in the number of flown kilometers. Due to its greater distance, the segments of the route from Trondheim to Gran Canaria are anticipated to show a larger total value for contrail ATR20 and SOC than, for example, the segments of the route from Istanbul to Madrid. Since we are not interested in the influence of route distance on our results, we again resort to specific contrail ATR20 as a parameter for analysis. Since this analysis also looks at the cost of the solutions, specific SOC will be used as well. A decrease in $\overline{ATR20}_{contrail}$ presumably indicates a decrease in overall $ATR20_{contrail}$. However, in theory, a decrease in $\overline{ATR20}_{contrail}$ could also be result of merely an increase in the number of flown kilometers. The same ambiguity holds for any changes in specific SOC ($\overline{SOC}$ $[\$/km]$). While presenting the results of this Pareto front analysis, we do not want to appreciate a flight for just increasing its flight distance, but only for decreasing its total contrail ATR20 and/or SOC. Hence, the contrail ATR20 and SOC of a flight segment are divided over the effective distance (d_e $[km]$) instead of the actual distance (d $[km]$). This effective distance is defined as the segment's distance component that is parallel to the GC line between the flight's airport of departure and airport of arrival. This is illustrated in figure 3.9. The effective segment length is calculated as the segment length multiplied with cosine of the angle between the segment and the GC line, θ $[-]$. Dividing the the contrail ATR20 and SOC over d_e introduces two new parameters: the effective specific contrail ATR20, or $\overline{eATR20}_{contrail}$

[2]https://pypi.org/project/suntimes/, Suntimes 1.1.2, Paul Mathis, Accessed: November 3, 2022

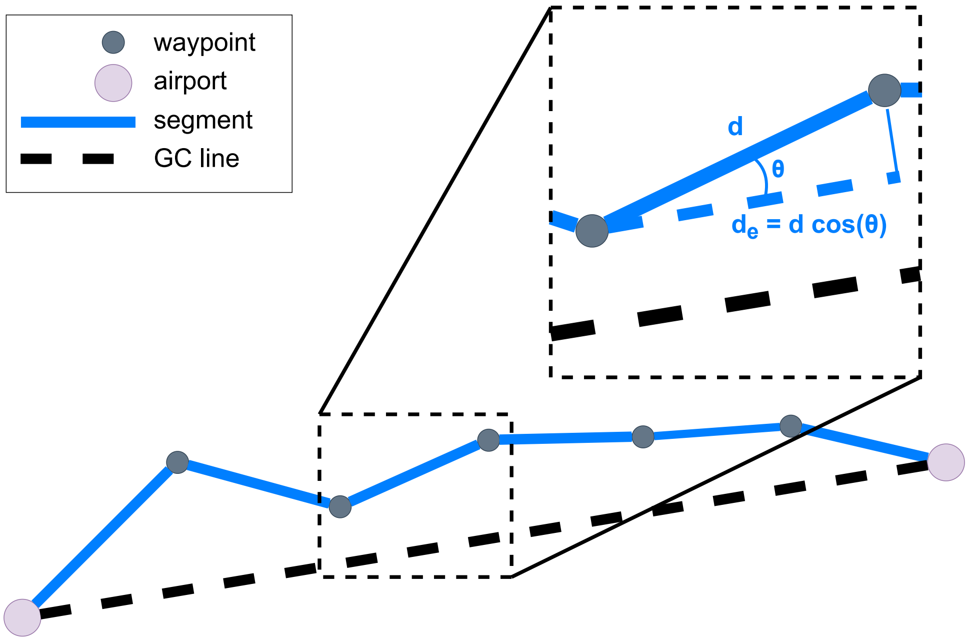

Figure 3.9: Calculation of the effective segment length, d_e, from the segment length, d, and the angle between the segment and the GC line, θ.

$[K/km]$ and the effective specific SOC, or $\overline{eSOC}$ $[\$/km]$. By using $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$, the Pareto efficiency of a flight is not clouded by a change in actual flight distance. We note that within the freedom of the trajectory optimisation (see figure 3.3), no negative value for d_e can occur.

The classification of a segment as nighttime or daytime segment is identical to that found under the header “Contrail ATR20: Waypoint Data”. For each of the three waypoint data solutions the mean $\overline{eATR20}_{contrail}$ and mean $\overline{eSOC}$ of the segments is calculated separately for the daytime and nighttime segments. This is done by dividing the sum of $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$ for all day segments over the number of day segments. Naturally, the sum of $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$ for all night segments is divided over the number of night segments. As this analysis focuses on mitigation, we will plot the change in mean $\overline{eATR20}_{contrail}$ and mean $\overline{eSOC}$ w.r.t. cost optimal solution. These values will be plotted to form four Pareto fronts: one for winter day, one for winter night, one for summer day and one for summer night. While the emphasis will be with the difference between daytime and nighttime, the generated plot will thus additionally show seasonal effects.

Average Pareto Fronts - Flight Data

The third analysis approach is comparable to that of the previous analysis. Again the focus of the analysis is on the day-night difference in mitigation potential and eco-efficiency. However, the data used for this analysis is the flight data.

Keeping in mind the objective of this thesis, we will again make a distinction between daytime and nighttime. However, opposed to the previous analysis approaches, no distinction between day and night can be made on segment level, since the flight data provides information only on the level of a complete flight. Consequently, a complete flight is characterised as either a daytime flight or a nighttime flight. The day-night classification of a flight is based on an approximation of its halfway point and halfway timestamp. A flight’s halfway point is obtained by averaging the departure and arrival coordinates and height. The halfway timestamp is calculated from the waypoint data for the cost optimal solution. For the corresponding flight in the waypoint data, the average of the timestamp of departure and the timestamp of arrival is calculated. When the halfway location and timestamp are found, the time of sunrise and sunset for the halfway location are collected with the help of the python suntimes model. Finally, again the procedure of figure 3.7 will be followed to classify a flight as either a daytime or a nighttime flight.

Just as for the waypoint data, $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$ will be used towards the resulting Pareto front. To arrive at these effective, specific values, the total contrail ATR20 and SOC of a flight are divided by the great circle distance between the airport of departure and the airport of arrival, and not by the actual flight distance. Contrarily to the previous waypoint data analysis, no calculation such as that of θ and the segment d_e are required for the effective values.

Before combining all solutions into one average Pareto front, the change in $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$ with respect to the cost optimal solution is calculated for each solution, indexed with i, of every flight, indexed with f. The change in effective, specific contrail ATR20 and SOC of each solution ($\Delta\overline{eATR20}_{contrail,f,i}$ and $\Delta\overline{eSOC}_{f,i}$) is calculated with equation 3.13 and 3.14, respectively.

$$\Delta\overline{eATR20}_{contrail,fi} = \overline{eATR20}_{contrail,fi} - \overline{eATR20}_{contrail,f0} \tag{3.13}$$

$$\Delta\overline{eSOC}_{fi} = \overline{eSOC}_{fi} - \overline{eSOC}_{f0} \tag{3.14}$$

Next, all the separate Pareto fronts of $\Delta\overline{eSOC}_{fi}$ versus $\Delta\overline{eATR20}_{contrail,fi}$ are combined into a single daytime and a single nighttime Pareto front. Since the number of points along the Pareto front varies per flight, the flight data needs some further preparation before the Pareto fronts can be combined. The invented procedure for preparing and combining the Pareto fronts is explained via a step-wise example (see figure 3.10). To combine a Pareto front of 4 points with a Pareto front of 9 points, the amount of points are brought at an equal number. To not lose any data, 5 points will be added to the 4-point-front, rather than removing 5 points from the 9-point-front. The information of each of the existing 4 points is valued equally, hence they all receive the same budget to contribute to the generation of the new points. In this case we are making 5 extra points from 4 points, so the budget is 5/4 = 1.25 per original point. The budget of the first point (the cost optimal point) is spent partly to duplicate itself, after which a budget of 0.25 remains. This 0.25 budget is used to contribute to a new point, which is completed with 0.75 of the budget of the second point of the 4-point-front. This second point now has 0.5 of its budget left, which is not enough to duplicate itself, but is spent as a contribution to the next new point instead. This chain of duplication and interpolation continues until the budget of the fourth and last point is spent. The 4-point-front is now made into a new 9-point-front. As a last step the combined Pareto front is created by linear interpolation of each point pair of the two 9-point-fronts. NB: although the given example only features direct multiplication for the first and last point of the 4-point-front, duplication (or higher multiplications) can also occur for middle points.

The advantage of the method in figure 3.10 is that the cost optimal and contrail optimal point of the resulting Pareto front are exactly equal to the average of the cost optimal and contrail optimal points of all contributing Pareto fronts. However, the average value for intermediate points is more ambiguous. This is caused by the fact that intermediate points of the Pareto front of an individual flight are not necessary equally spaced. For example, consider the combination of two 3-point Pareto fronts as shown in figure 3.11. For the green front the intermediate point is almost equal to the contrail optimal point, for the blue front the intermediate point almost equals the cost optimal point. The green and blue dashed lines show a plausible shape for a continuous Pareto front, based on the three known points and previous research (e.g., Lührs et al. (2021)). The red dots show the merged Pareto front resulting from the method in figure 3.10. The pink dashed line shows the average of the green and blue dashed line. As can be seen, the intermediate point for the combined Pareto differs from the dashed line. Given the usually convex shape of the (theoretical) continuous Pareto font, this bias often

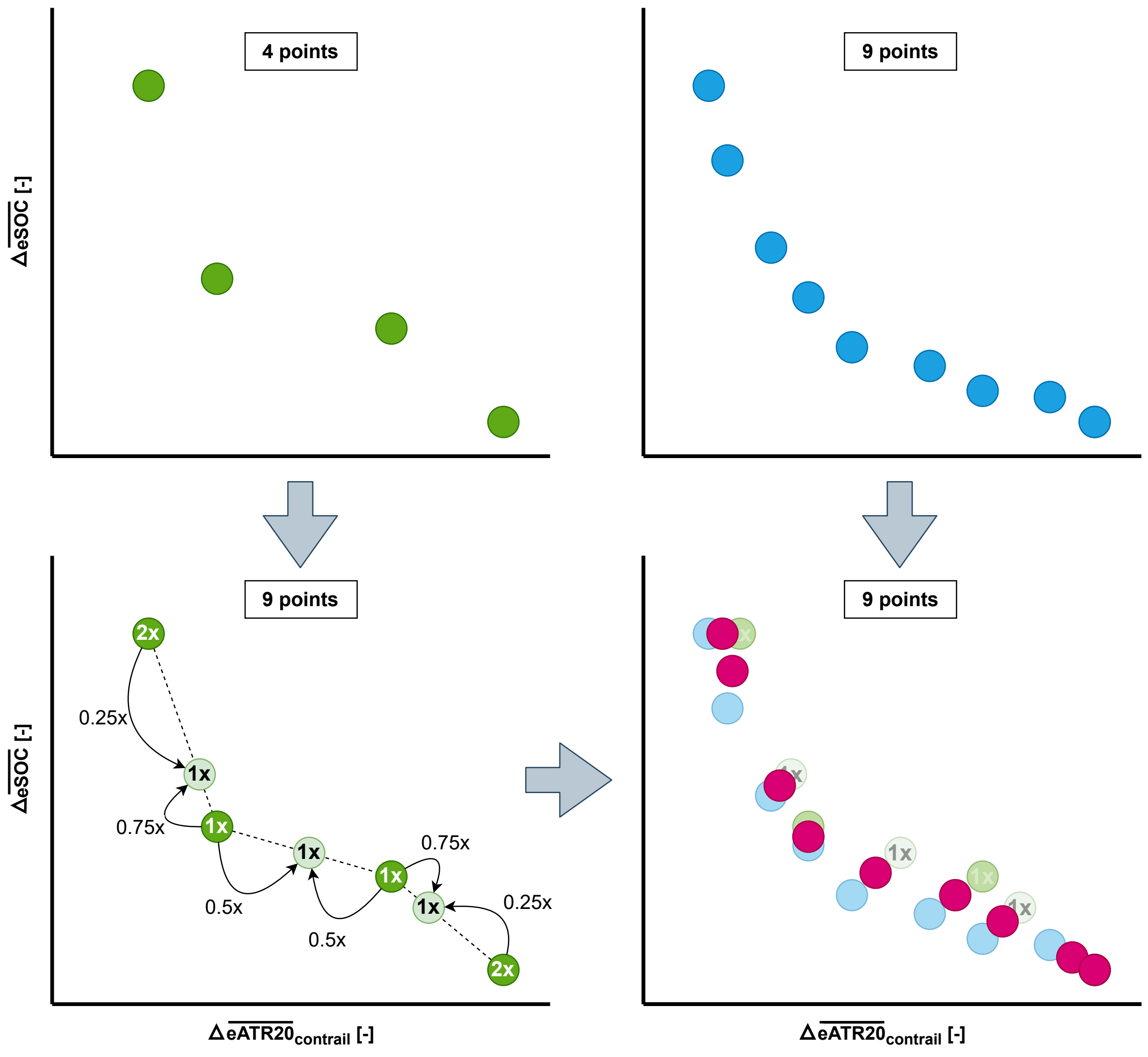
4 points
9 points
9 points
9 points
$\Delta\overline{eSOC}$ [-]
$\Delta\overline{eSOC}$ [-]
2x
0.25x
1x
0.75x
1x
1x
0.5x
1x
0.75x
1x
0.25x
0.5x
2x
$\Delta\overline{eATR20}_{contrail}$ [-]
$\Delta\overline{eATR20}_{contrail}$ [-]

Figure 3.10: Visualisation of how two Pareto fronts of different size are combined into one average Pareto front.

results a conservative estimation of the eco-efficiency of an intermediate solution. This will be taken into account within the discussion of the results.

Just as for the previous analyses, the average Pareto fronts will not only be plotted separately for daytime and nighttime, but also for winter and summer to reveal seasonal effects. Although the results of the average Pareto from waypoint data and the results from the average Pareto front from flight data are very similar in nature, they are not the same. The main difference being: for waypoint data distinction between day and night happens on segment level, for flight data this distinction happens on flight level. The other difference is found in the resolution of the Pareto front. The number of intermediate solutions between the SOC optimal and contrail optimal solution is generally higher for the flight data than for waypoint data. This results in a more well defined Pareto front for the flight data.

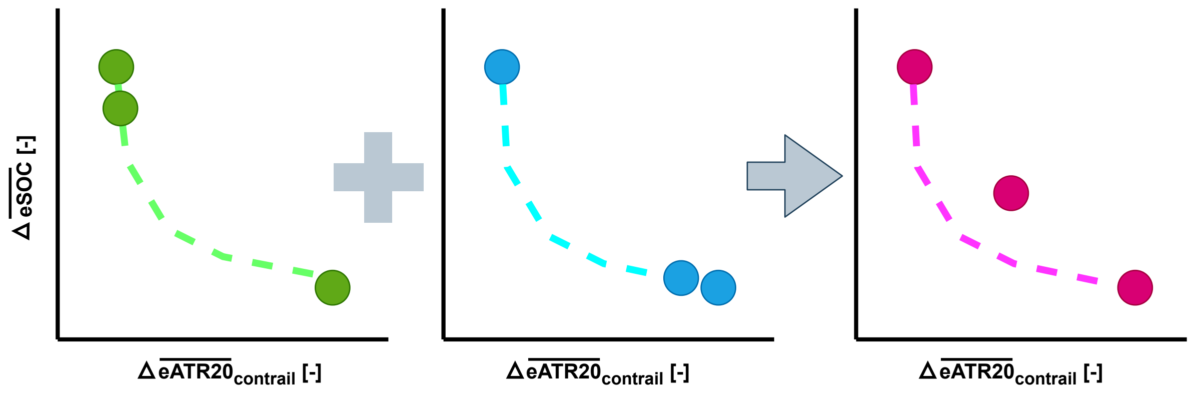
$\Delta\overline{\text{eSOC}}$ [-]
$\Delta\overline{\text{eATR20}}_{\text{contrail}}$ [-]
$\Delta\overline{\text{eATR20}}_{\text{contrail}}$ [-]
$\Delta\overline{\text{eATR20}}_{\text{contrail}}$ [-]

Figure 3.11: Visualisation of conservative bias for intermediate points along the Pareto front.

Average Pareto Front Excluding Extra Cooling - Waypoint Data

Studying opportunities for the mitigation of climate impact occasionally sparks an ethical discussion. Should we use our knowledge to purposely intervene in the Earth's climate system, e.g., to counteract climate change? Activities that do indeed purposely intervene in the climate system are often referred to as "geoengineering". While keeping this ethical debate on geoengineering in mind, the use of flight diversions to form extra cooling contrails is questionable. Although these diversions aim to reduce the net climate impact of aviation, the means comprises a local increase in the magnitude of aviation's climate impact, albeit in the form of negative RF. Moreover, trying to purposely induce a cooling contrail may have unforeseen consequences: e.g., the contrail could be warming due to wrong predictions. For this reason, the following analysis will attempt to exclude any extra cooling from the average Pareto fronts and evaluate how the results are affected. Besides achieving a result that shows the mitigation potential without geoengineering, the results of this analysis will also give insight in the dependency of contrail ATR20 mitigation on extra cooling.

The first step towards the exclusion of extra cooling is a definition of what extra cooling is. Here, the used definition makes use of the cost optimal trajectory as a reference. A flight segment of a more contrail optimal trajectory is said to be extra cooling if: (1) the new segment forms a contrail with negative contrail ATR20 and (2) the corresponding segment of the reference trajectory has a higher contrail ATR20 or no contrail ATR20. Now, a strategy for the exclusion of extra cooling contrails needs to be formulated. According to the definition above, this strategy requires a definition of 'a flight segment' and 'corresponding segment of the reference trajectory'. Furthermore, we need to find a way to deal with the flight disruption, if a segment is indeed removed. The exclusion of extra cooling, or even any cooling, from optimised trajectories is not found in any preexisting literature. Hence, a strategy is formulated below.

The input data for the removal of extra cooling will be that of waypoint data. We thus have a SOC optimal solution, $\sim$0.5% extra SOC solution and the contrail optimal solution. As described above, 'flight segment' should be defined. For this we will copy the definition of Yamashita et al. (2020), where a trajectory is built from 100 flight segment. For each flight segment the 'corresponding segment of the reference trajectory' will be the segment of the SOC optimal solution with the same index. So for example, the 0th segment of the SOC optimal solution is the corresponding segment for the 0th segment of the contrail optimal solution. The 1st SOC optimal segment corresponds to the 1st contrail optimal segment, and so on. When, according to the given definition, a segment of the $\sim$0.5% extra SOC solution or the contrail optimal solution is identified as 'extra cooling', the segment is removed.

The exclusion of extra cooling segments of a trajectory results in gaps in said trajectory. To still arrive at a realistic overall image of the contrail ATR20 and SOC, we will use a replacement tactic. For the '∼0.5% extra SOC' solution excluded segments will be replaced by corresponding segments of the SOC optimal solution. Excluded segments of the contrail optimal solution are replaced by segments from the '∼0.5% extra SOC' solution. In the case the latter segments were already replaced, the segment in question is replaced by corresponding SOC optimal segment as well.

The replacement of segments that feature extra cooling is illustrated in figure 3.12. Walking through the figure from left to right, we see the following for the '∼0.5% extra SOC' solution: the first two segments are both warming segments and are thus not replaced. The third and fourth segment are both cooling and more so than than the SOC optimal segments. Corresponding to the definition of extra cooling, these two segments are replaced by the SOC optimal segments. The last two segments are also cooling, but not more than the SOC optimal segments did. Hence, these segments are allowed to stay part of the '∼0.5% extra SOC' solution. For the contrail optimal solution the first and fourth segment are included, as they are not cooling. However, the second, third and fifth segment do meet the criteria for extra cooling and are indeed replaced. The second and fifth segment shall be replaced by the corresponding segments of the '∼0.5% extra SOC' solution, as those segments were kept after their replacement tests. However, the third segment will have to use the third SOC optimal segment as a replacement. The last segment is cooling, but not with a larger magnitude than the segment of the SOC optimal solution and can thus remain included.

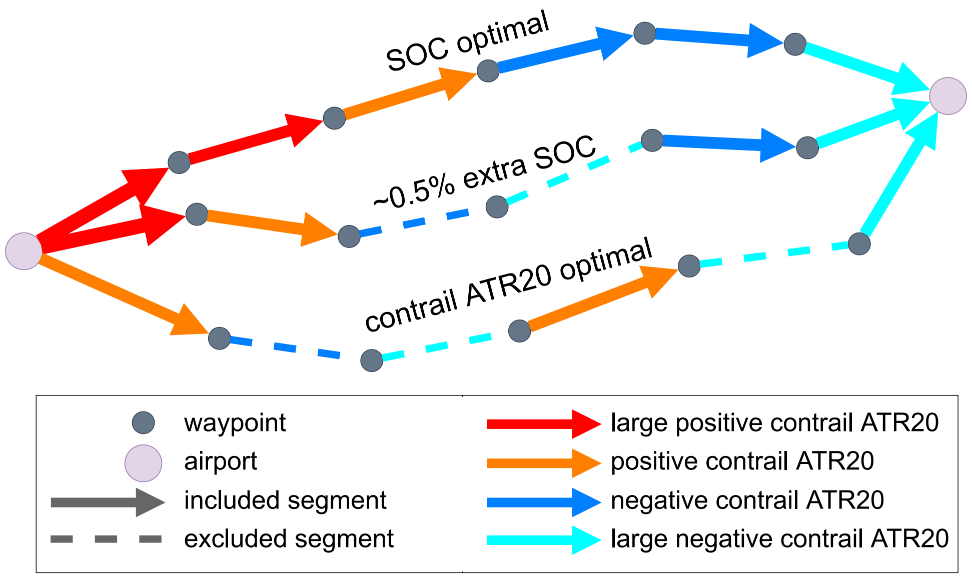

Figure 3.12: Illustration of the exclusion of segments with extra cooling.

The rest of the analysis for excluded extra cooling is identical to the plain Pareto front analysis for waypoint data.

3.3.3 Data Analysis: Variability with the Threshold Time Until Sunrise

The next phase of the analysis focuses on the variability of the results of the previous Section 3.3.2 with $TH_{sunrise}$. The resulting analysis should fulfil objective two of this thesis. As discussed in Section 3.2.3, the analysis will not only test the robustness w.r.t. $TH_{sunrise}$ but also give an indication of the sensitivity to assumed contrail lifetime.

The set up of this analysis phase is straightforward, as it mainly entails a repetition of the analyses of Section 3.3.2 for the different $TH_{sunrise}$ selected in Section 3.2.3. To clearly show the variability, the results for each different $TH_{sunrise}$ will be plotted alongside each other in the same figure. During the analysis, the postprocessing method will account for the $TH_{sunrise}$ that was used in the simulation. This means that the daytime-nighttime classification of segments (for waypoint data) or flights (for flight data) during the postprocessing will be identical to the classification of the ACCF submodel during the simulation. Moving from a $TH_{sunrise}$ of 0 hours to 3 or 6 hours, this results in a greater portion of segments and flights being classified as daytime both during the simulation and the postprocessing.

3.3.4 Data Analysis: Search for New Pareto Front

The last analysis for this thesis aims to complete the third thesis objective and therefore explores a climate optimal practice to spend a certain budget of SOC on the minimisation of contrail ATR20. This exploration makes use of the difference between daytime and nighttime for contrail ATR20 and is approached as described below.

Using the flight data, an average Pareto front can be formed from the average daytime Pareto front and the average nighttime Pareto front. The points of the average Pareto front are found by averaging $\Delta\overline{eATR20}_{contrail}$ for two points that have an equal $\Delta\overline{eSOC}$. However, the points of the two Pareto fronts are unlikely to contain any points with exactly equal values for $\Delta\overline{eSOC}$. Thus, to find such points, either the daytime or the nighttime Pareto front is resampled through linear interpolation. Additionally, the contrail optimal point of the daytime average Pareto front is compared to the contrail optimal point of the nighttime average Pareto front. If this daytime point has a larger value for $\Delta\overline{eSOC}$ than the nighttime point, the transcending daytime points are averaged with the nighttime contrail optimal point. If the daytime contrail optimal point has a larger value for $\Delta\overline{eSOC}$ than the nighttime contrail optimal point, the reverse holds. The resulting average points show the value $\Delta\overline{eATR20}_{contrail}$ if an equal amount of $\Delta\overline{eSOC}$ is spend during both day and night.

The next steps show what would happen if we calculate the average $\Delta\overline{eATR20}_{contrail}$ of a daytime and a nighttime point that do not have an equal $\Delta\overline{eSOC}$. In pursuit of new average points we calculate the average of every possible combination of a point on the daytime Pareto front and the nighttime Pareto front. The newly found points are collected together with the points of the earlier found average Pareto front. Also the two parent points of each new point are stored in memory. Next, the collection of points is run through an algorithm that identifies the Pareto efficient points[3]. By comparison with the average Pareto front and comparison between themselves, the new Pareto efficient points are identified. An example of the search for a single new point is illustrated in figure 3.13. This analysis should indicate how using the difference between daytime and nighttime can be used to achieve the largest possible $\Delta\overline{eATR20}_{contrail}$ at a certain $\Delta\overline{eSOC}$. NB: the results of this analysis only apply to fleets that operate 50% of the flights during daytime and 50% during nighttime, or flights that spend half the trajectory in daylight and half in darkness. Other ratios require that a weight is applied in generating the average and new points in the Pareto plot.

[3]`https://stackoverflow.com/questions/32791911/fast-calculation-of-pareto-front-in-python`, Fast calculation of Pareto front in Python, Accessed: November 3, 2022

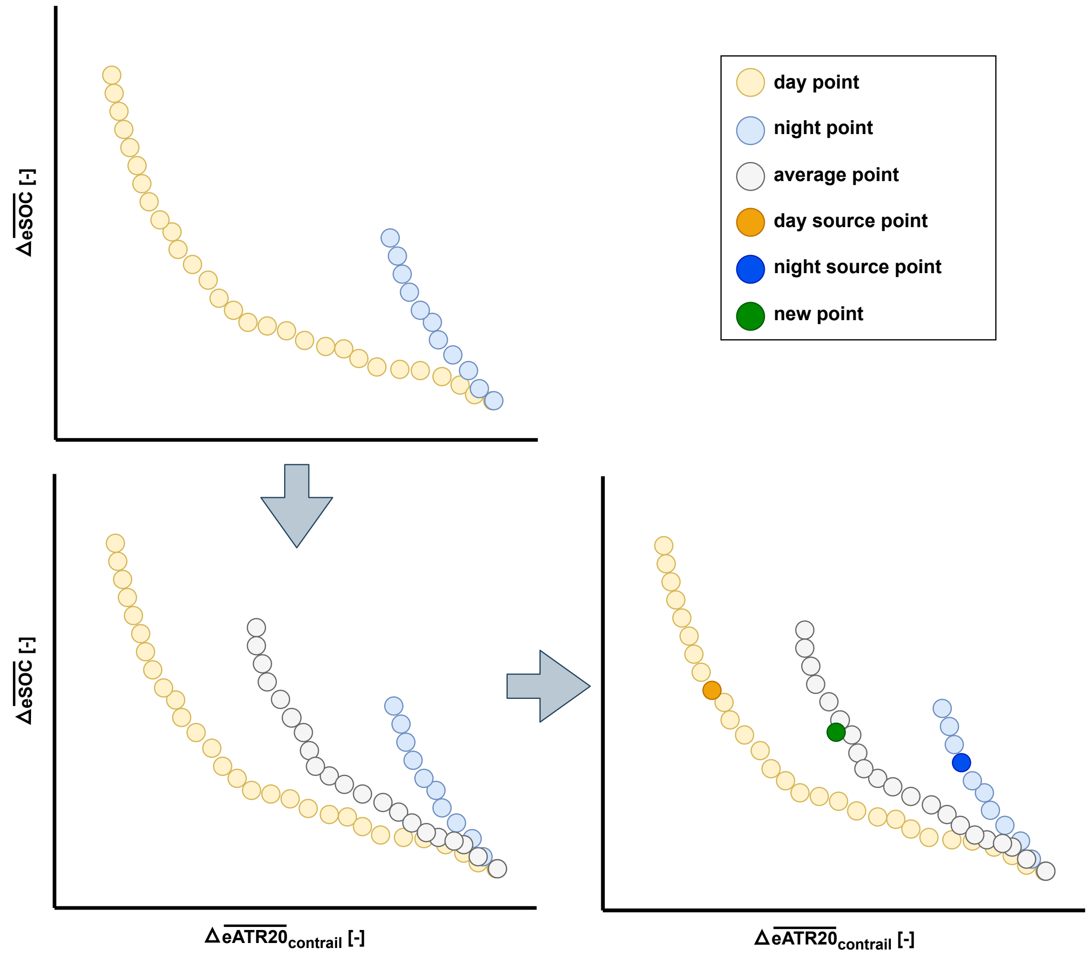
day point
night point
average point
day source point
night source point
new point
$\Delta\overline{eSOC}$ [-]
$\Delta\overline{eSOC}$ [-]
$\Delta\overline{eATR20}_{contrail}$ [-]
$\Delta\overline{eATR20}_{contrail}$ [-]

Figure 3.13: Illustration of the search for a new point (green) that achieves an improved $\Delta\overline{eATR20}_{contrail}$ at a certain $\Delta\overline{eSOC}$. The new point is the average of two source points (orange and dark blue) found among the daytime (yellow) and nighttime (light blue) points.

Chapter 4

Results

This chapter presents the results of the analyses that were described in the previous chapter. For the presentation of the results, we keep the same structure as in Section 3.3. Firstly, daytime contrail climate impact and its mitigation is compared to nighttime impact and mitigation in Section 4.1. Subsequently, the variability of the results with a change in threshold time until sunrise is discussed in Section 4.2. Lastly, newly found points within the flight data Pareto plot, making use of the difference between daytime and nighttime, are presented in Section 4.3. Please note that this chapter focuses on the presentation of the results and aims to describe their particularities. Chapter 6, will be used to interpret the described observations.

4.1 Comparing Daytime and Nighttime Contrail Climate Impact

This section presents the results that answer to the first research objective of this thesis. The methods of analysis for these results were described in Section 3.3.2. As described in that section, the difference between daytime and nighttime is first analysed for the baseline, i.e., the cost optimal solution in Section 4.1.1. Then, shifting the focus to mitigation efforts, the difference between daytime and nighttime is studied using the more contrail optimal solutions in Section 4.1.2. The results of Section 4.1.2 are based on both waypoint data and flight data. For waypoint data only, Section 4.1.3 shows how much the mitigation of contrail ATR20 depends on extra cooling.

4.1.1 Cost Optimal Day and Night Specific Contrail ATR20

Figure 4.1 shows the specific day and night contrail ATR20 for SOC-optimised flights as found from waypoint data. The bars present the seasonal average for $\overline{ATR20}_{contrail}$ in winter and summer separately. The whiskers show the range of a day to day single standard deviation from the mean. What we first see in this figure is that, for the cost optimal solution, the difference in mean $\overline{ATR20}_{contrail}$ between daytime and nighttime is marginal, certainly in the perspective of the standard deviation. In winter, the mean daytime $\overline{ATR20}_{contrail}$ shows a slightly larger value than the nighttime $\overline{ATR20}_{contrail}$. For summer, again the difference between day and night is negligible, but for this season the mean nighttime $\overline{ATR20}_{contrail}$ shows a slightly higher value than that of daytime.

Whereas the figure shows little difference between day and night, a clear difference is observed between the two seasons. For both day and night the summer $\overline{ATR20}_{contrail}$ has nearly four times the magnitude that we observe for the winter season. The difference in magnitude can be partly explained through the data presented in figure 4.2. To create figure 4.2 (a,b and c), first, the mean and standard deviation of flight altitudes were calculated to find which hybrid pressure levels of the EMAC T42L31ECMWF grid correspond to the airspace that is used by the cost optimal flights. At these hybrid pressure levels, the mean values for *T*, *OLR* and *potcov* over Europe are found for the days that were used for the simulations of this thesis. The mean values for *T*, *OLR* and *potcov* are obtained from the base model ECHAM5 and submodels RAD (Dietmüller et al. (2016)) and CONRAIL, respectively. Moreover, the day to day standard

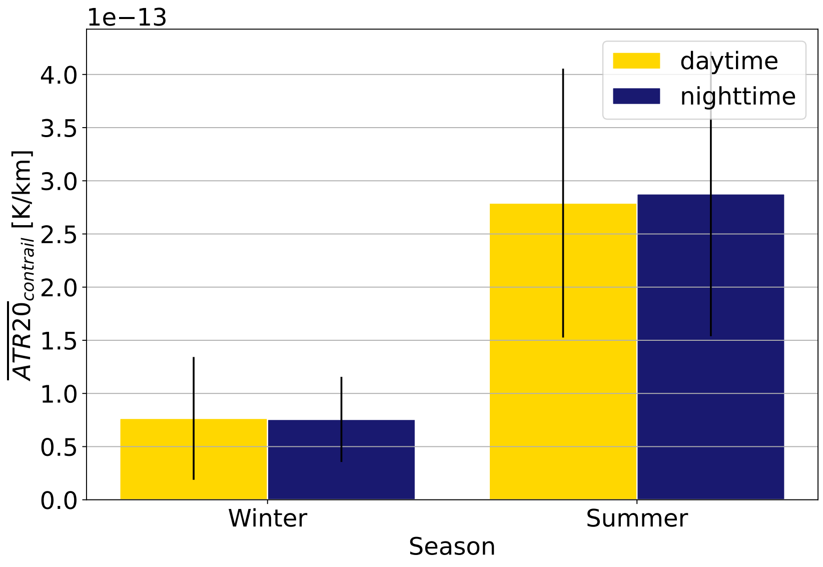

Figure 4.1: Mean, specific, SOC optimal day and night contrail ATR20 for winter and summer, $TH_{sunrise}$ = 0 *h*. Whiskers represent the day to day standard deviation.

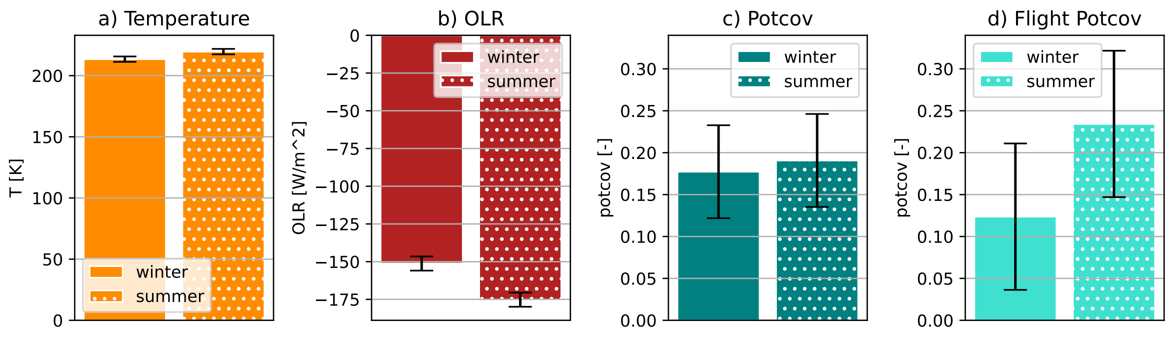

Figure 4.2: The mean values for (a) *T*, (b) OLR and (c) *potcov* in winter and summer at the SOC optimal flight levels. (d) Shows the mean *potcov* as actually experienced by the flights. Whiskers represent the day to day standard deviation.

deviation for these parameters is extracted. Additionally, the mean values for *potcov* actually experienced by the flights (flight *potcov*) are also found from the AirTraf output data.

Figure 4.2d presents a large flight *potcov* for the summer flight segments, which is on average around 1.9 times larger than in winter for the performed simulation. That the flight *potcov* is largest for summer can be traced back to figure 4.2c, where summer *potcov* also shows to be larger than winter *potcov*. This is possibly caused by the position of the tropopause, that changes along with the seasonal cycle, and the relatively high flight altitude of the simulated cost optimal flights. Castino et al. (2022), who also used AirTraf and EMAC, already observed that a large part of SOC optimal winter flights cruises above the tropopause. A large part of air traffic is thus situated within the stratosphere, which is known to be too dry for persistent contrail formation (Fichter et al. (2005)). In summer, the average altitude of the tropopause is increased, leading to a higher fraction of air traffic flying within the upper troposphere, which features more favourable conditions for the formation of persistent contrails (Fichter et al. (2005)). This may explain the higher summer *potcov* in figure 4.2c. Seemingly, the spacial variability of *potcov* allowed for an even larger difference between winter and summer flight *potcov*. Further inspection of the contrail aCCF expression (equation 3.3) explains the rest of the magnitude difference: European airspace features higher temperatures in winter than in summer. Furthermore, the summer OLR has a larger magnitude than the winter OLR, since the Earth surface has a higher temperature in summer.

Considering figure 4.1 as a starting position for the mitigation of contrail ATR20, we could conclude that the mitigation potential is largest in summer. In summer, the $\overline{ATR20}_{contrail}$ is large to begin with, while this is significantly smaller in winter. Assuming that avoiding warming contrails or creating extra cooling contrails takes equal effort in winter and summer (which is not necessarily true), trajectory optimisation in summer should be more rewarding in terms of absolute numbers. Given the small difference between daytime and nighttime, figure 4.1 does not allow for conclusions about the difference between daytime and nighttime mitigation.

4.1.2 Day and Night Mitigation of Contrail ATR20

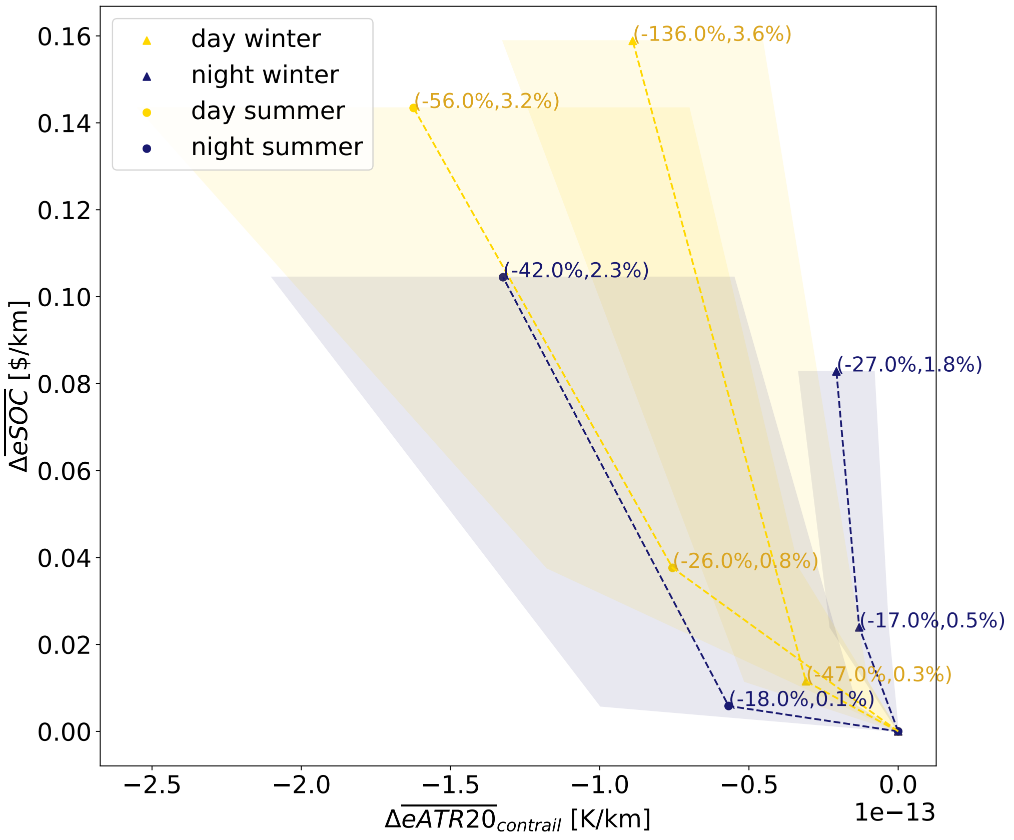

Figure 4.3: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter and summer generated with waypoint data, $TH_{sunrise}$ = 0 *h*. The percentages placed at the contrail optimal solution indicate the relative differences to the SOC optimal solution. Shaded areas show the day to day standard deviation of $\Delta\overline{eATR20}_{contrail}$.

Figure 4.3 shows the mean day and night, winter and summer Pareto fronts, as obtained from waypoint data. For all points but the cost optimal point, the percentage change in $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$ with respect to the cost optimal point is displayed between brackets. The shaded areas show the day to day standard deviation of $\Delta\overline{eATR20}_{contrail}$. For winter and summer, figure 4.3 shows that the maximum potential to mitigate contrail ATR20 is higher during daytime than during nighttime. This holds both in terms of absolute and relative numbers. Note that these higher daytime maxima do come at the expense of a higher increase in SOC. In winter, the maximum decrease in $\overline{eATR20}_{contrail}$ even reaches over 4 times the maximum decrease during nighttime. Moreover, away from the contrail optimal point, the mean winter daytime mitigation is achieved at a lower SOC than winter nighttime mitigation. Or alternatively described: on average, at the same increase in $\overline{eSOC}$ daytime winter mitigation allows a larger decrease in $\overline{eATR20}_{contrail}$ than nighttime winter mitigation. Regarding the variability of the winter fronts, this finding appears to be robust.

In summer, the difference between daytime and nighttime mitigation is less pronounced than in winter. Starting from the cost optimal solution, the mean values in figure 4.3 indicate that initial mitigation efforts are more cost-effective for nighttime than for daytime. However, this could also be the result of the limited number of points along the waypoint Pareto front. A higher sample frequency might show that, at low cost, summer daytime and nighttime mitigation are equally eco-efficient. Nonetheless, when a larger reduction of $\overline{eATR20}_{contrail}$ is desired, the mean $\overline{eSOC}$ spent on nighttime mitigation approaches the amount spent on daytime mitigation. The summer day to day variability of $\Delta\overline{eATR20}_{contrail}$ is large, making it hard to draw robust conclusions about the relative position of the summer daytime and nighttime front.

Another deduction from figure 4.3 is that, in absolute terms, the summer daytime and nighttime contrail optimal solution show a larger reduction of $\overline{eATR20}_{contrail}$ than the winter contrail optimal solution. This is possibly explained by the observation in Section 4.1.1 that the summer specific contrail ATR20 for the SOC optimal solution is larger to begin with. While summer shows the highest absolute mitigation, the highest relative mitigation is only observed for summer nighttime and not for summer daytime. The -56% change in $\overline{eATR20}_{contrail}$ for summer daytime cannot match the -136% change observed for winter daytime. Summer night mitigation shows to be more cost-effective overall compared to winter night mitigation. For daytime, winter mitigation appears cheaper than summer mitigation in the low-cost region (the part of the Pareto front where a low amount of extra $\overline{eSOC}$ is spend to reduce $\overline{eATR20}_{contrail}$). If a higher amount of extra $\overline{eSOC}$ is spend, daytime mitigation is cheapest in summer.

Please note that figure 4.3 shows that winter daytime mitigation can result in an overall cooling impact, indicated by the -136% change in effective specific contrail ATR20. This indicates that flights during winter daytime, more than during summer daytime, make use of cooling contrails to lower the $\overline{eATR20}_{contrail}$. Possibly, this also explains why winter mitigation shows a large difference between the daytime and nighttime Pareto front, while the two summer fronts are found fairly close to each other.

Figure 4.4 plots the average day and night, winter and summer Pareto fronts for flight data. Just as for figure 4.3, each contrail optimal point of a Pareto front is accompanied by the percentage change with respect to the cost optimal point. Although figure 4.3 and figure 4.4 are similar in nature, it is important to note that the two figures present different data. Whereas figure 4.3 distincts daytime from nighttime per segment, figure 4.4 only makes this distinction per flight, even though these flights may consist of both daytime and nighttime segments (see section 2.1.7). The nature of flight data makes it impossible to calculate a day to day standard deviation. Hence, a representation of the variability is omitted for figure 4.4.

From figure 4.4 is immediately visible that the mitigation of $\overline{eATR20}_{contrail}$ is more cost-effective for daytime flights than for nighttime flights, i.e., the daytime Pareto fronts are found to the lower left of the nighttime Pareto fronts. Moreover, the maximum achievable reduction in $\overline{eATR20}_{contrail}$ is larger for daytime flights than for nighttime flights, both in absolute and a relative terms. However, this also comes at a higher increase in SOC.

While comparing the two seasons, we observe that in summer $\overline{eATR20}_{contrail}$ is mitigated more efficiently than in winter when considering absolute values. As earlier described and

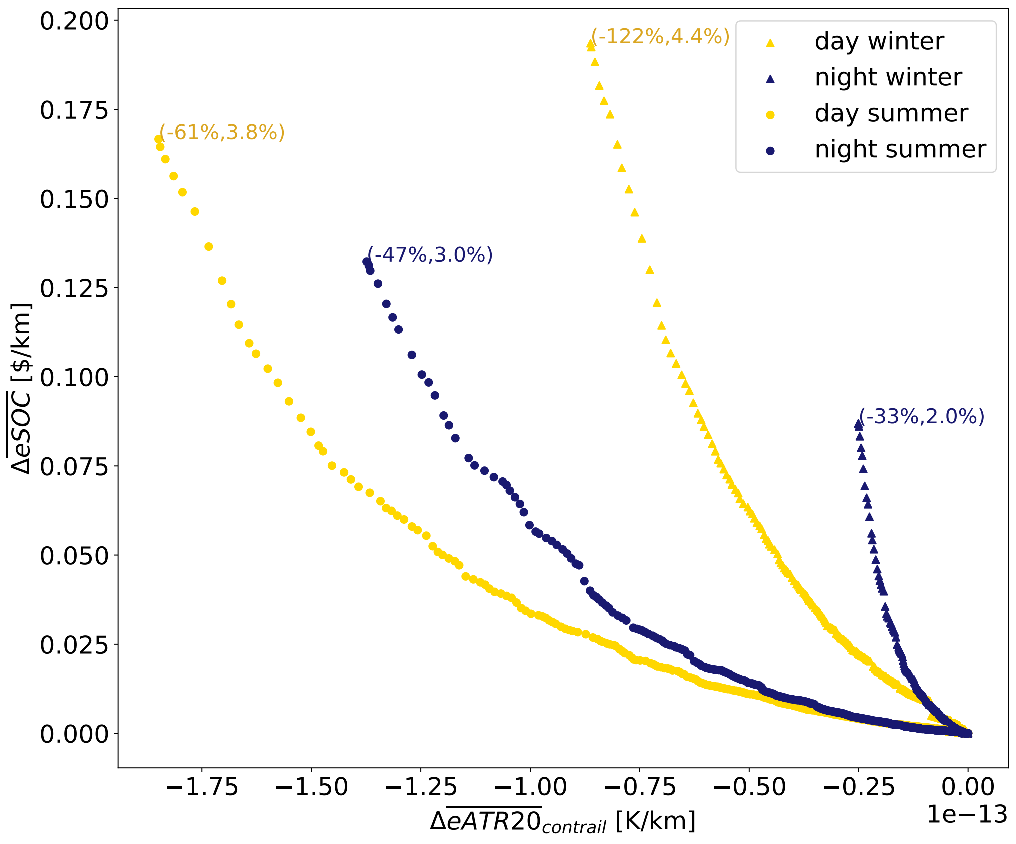

Figure 4.4: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter and summer generated with flight data, $TH_{sunrise} = 0\ h$. The percentages placed at the end-points of the fronts indicate the change w.r.t. the SOC optimal solution.

confirmed by the waypoint data, this difference is possibly caused by the large magnitude that we observed for summer $\overline{ATR20}_{contrail}$ in figure 4.1. Just as in figure 4.3, in relative terms, the daytime winter Pareto front performs about a factor two better than in the summer. Also for nighttime the ratio between the winter and summer contrail optimal point of Figure 4.4 is in accordance with figure 4.3.

Another similarity to figure 4.3 is that the difference between day and night mitigation is larger for winter than for summer. This once more indicates that winter daytime mitigation makes relatively much use of the formation of extra cooling contrails, since the option to create extra cooling contrails is a key difference between daytime and nighttime mitigation.

4.1.3 Mitigation of Contrail ATR20 Without Extra Cooling

Figure 4.5 duplicates the results of figure 4.3, except that any segments that showed extra cooling with respect to the corresponding cost optimal segment is replaced by a segment without extra cooling. The previous section already gave an indication to what extent extra cooling contrails are used for the mitigation of $\overline{eATR20}_{contrail}$. This section will show to what extent these indications are true.

As expected, replacing segments featuring extra cooling contrails mainly affects the daytime Pareto fronts, as cooling contrails cannot occur during nighttime. Clearly, the daytime

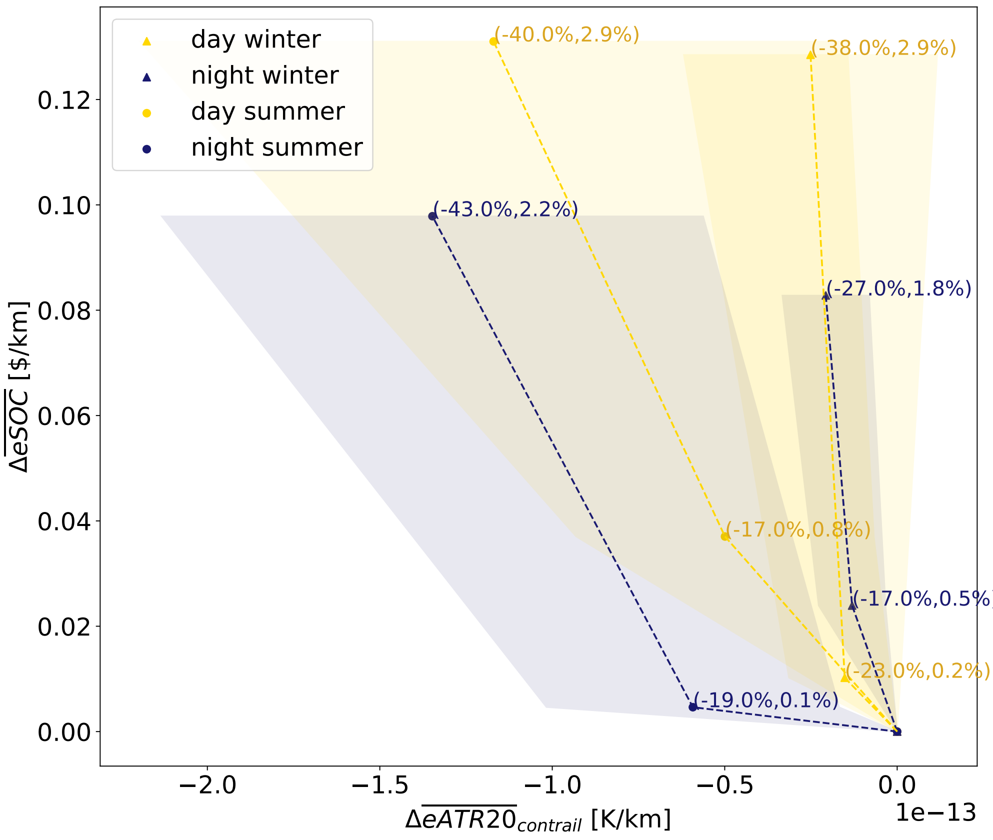

Figure 4.5: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter and summer generated with waypoint data, $TH_{sunrise} = 0\ h$. Extra cooling is excluded. The percentages placed at the end-points of the fronts indicate the change w.r.t. the SOC optimal solution. Shaded areas show the day to day standard deviation of $\Delta\overline{eATR20}_{contrail}$.

fronts face a reduction in their $\overline{eATR20}_{contrail}$ mitigation potential. However, the extent of the daytime mitigation potential reduction differs between winter and summer. Whereas summer daytime mitigation only loses 16% of its maximum relative mitigation potential, winter daytime mitigation has to give up 98%. The difference shows that summer daytime mitigation efforts are less dependent on the formation of extra cooling contrails than mitigation efforts in winter.

Furthermore, we observe that, over the complete spectrum, summer nighttime mitigation now achieves a higher $\overline{eATR20}_{contrail}$ reduction than daytime mitigation. In winter, the day-night difference that we have seen in figure 4.3 has shrunk considerably. The remaining daytime mitigation keeps a negligible advantage over nighttime mitigation. In particular for winter, the contrail optimal point of the Pareto appears to be more affected than the point of $\sim$0.5% extra SOC. This possibly indicates that initial (low $\Delta\overline{eSOC}$) mitigation is less depending on extra cooling than the mitigation efforts that lead to the contrail optimal point.

A close look at figure 4.5 shows that the summer nighttime front is affected too by the exclusion of extra cooling segments. The figure namely shows a slight increase in summer nighttime mitigation potential. This change is remarkable, given that nighttime segments cannot feature any cooling and are thus unaffected by the replacement procedure. That a shift occurred anyhow, can be explained by a number of daytime segments, that has been replaced by nighttime segments as a result of the replacement procedure. These extra nighttime segments affect the average nighttime $\overline{eATR20}_{contrail}$ and $\overline{eSOC}$). Table 4.1 displays the number of extra nighttime segments for the waypoint Pareto points without extra cooling with respect to the waypoint Pareto points that allowed extra cooling. Table 4.1 also shows that the number of nighttime segments did not change for the winter season, hence the winter nighttime Pareto front in figure 4.5 is identical to the nighttime Pareto front in figure 4.3.

Table 4.1: Change in number of nighttime segments between the waypoint Pareto points and the waypoint Pareto points without extra cooling.

Solution	Winter	Summer
SOC optimal	0	0
$\sim$0.5% extra SOC	0	+324 (+2.1%)
contrail optimal	0	+661 (+4.3%)

4.2 Variability of Results with Threshold Time Until Sunrise

This section presents the results that answer to the second research objective of this thesis. The methods of analysis for these results were described in Section 3.3.3. Following the approach of that section, the robustness of the results of Section 4.1 is tested against a changing $TH_{sunrise}$. For this purpose $TH_{sunrise}$ was altered from the original 0h to 3h and 6h. The variability of the bar plot results are shown in Section 4.2.1. The variation of the Pareto fronts is displayed in Section 4.2.2. In Section 4.2.3, the same is done for the waypoint Pareto fronts that exclude extra cooling.

4.2.1 Variability of Cost Optimal Day and Night Specific Contrail ATR20

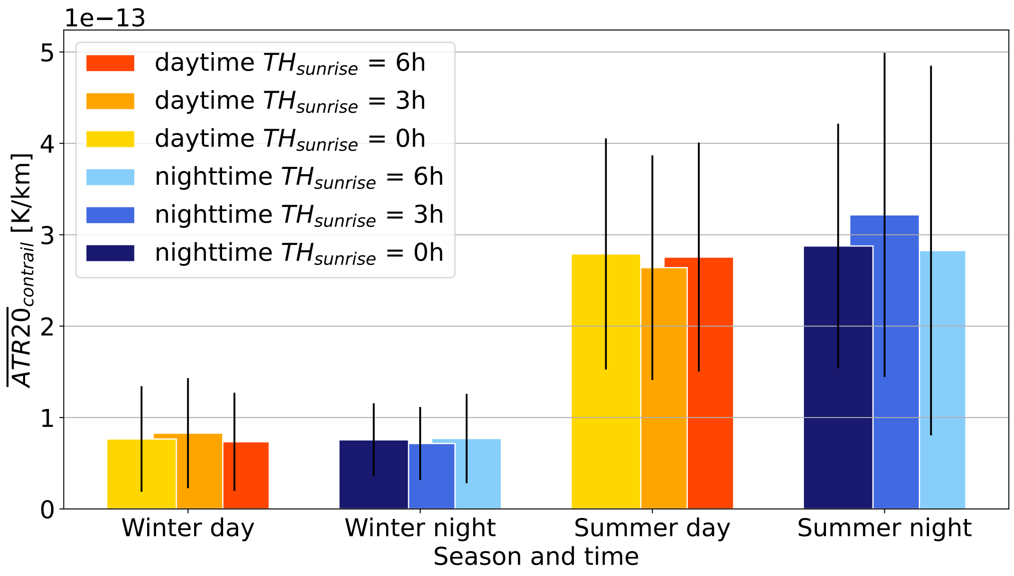

Figure 4.6: Specific, cost optimal day and night contrail ATR20 for winter and summer, for $TH_{sunrise}$ = 0, 3 and 6h.

Following the logic of figure 3.7 in Section 3.2.3, the main effect of an increase in $TH_{sunrise}$ is an increase in the number of flight segments that are treated as a daytime segment by the ACCF submodel. For $TH_{sunrise} > 0$, not only contrails formed during daytime are daytime segments, but also those that are formed during nighttime but within $TH_{sunrise}$ hours before sunrise. The resulting variability is first analysed by inspecting how the bar plot of figure 4.1 is affected by the increasing $TH_{sunrise}$. The bars for the different values of $TH_{sunrise}$ are plotted in figure 4.6. Moreover, the day to day standard deviation is shown by the whiskers.

Figure 4.1 shows that increasing $TH_{sunrise}$ has an equivocal effect on the specific contrail ATR20 of the simulated flights. In winter, changing from $TH_{sunrise}$ = 0h to $TH_{sunrise}$ = 3h, leads to a slightly increased daytime $\overline{ATR20}_{contrail}$ and a slightly decreased nighttime $\overline{ATR20}_{contrail}$. An opposite effect can be observed in summer, with the note that for summer nighttime the change in $\overline{ATR20}_{contrail}$ is more substantial than for the other three bars for $TH_{sunrise}$ = 3h. Although the changes are not large (all below 12%). Lastly, we observe that daytime and nighttime $\overline{ATR20}_{contrail}$ are no longer as equal in magnitude as shown for $TH_{sunrise}$ = 0h.

If we now inspect the bars corresponding to $TH_{sunrise}$ = 6h, we see that their values are almost equal to that of $TH_{sunrise}$ = 0h. Thus, the changes observed going from $TH_{sunrise}$ = 0h to $TH_{sunrise}$ = 3h are not escalated by a further increase of $TH_{sunrise}$.

As described, the increase of $TH_{sunrise}$ results in an addition of segments to the number of daytime segments and a reduction in nighttime segments. For winter, the results of figure 4.6 seem to indicate that flight segments occurring within 3 hours before sunrise have a relatively high average contrail ATR20. Segments situated between 6 and 3 hours before sunrise have a relatively low contrail ATR20, as the winter results for $TH_{sunrise}$ = 6h show that they offset the change seen for $TH_{sunrise}$ = 3h. For summer flight segments, the opposite conclusion is drawn. Overall, changes in the cost optimal $\overline{ATR20}_{contrail}$ are not large and not escalating, leading to the conclusion that the cost optimal $\overline{ATR20}_{contrail}$ is robust with respect to a varying $TH_{sunrise}$.

4.2.2 Variability of Day and Night Mitigation of Contrail ATR20

Next, for winter and summer, respectively, figure 4.7 and 4.8 show how the waypoint data Pareto fronts for daytime and nighttime vary with $TH_{sunrise}$. Experience from Section 4.1 tells us that a higher cost optimal $\overline{ATR20}_{contrail}$ results in a higher mitigation potential. For winter, the daytime front of figure 4.7 shows a variability that behaves according to this experience. For $TH_{sunrise}$ = 3h, the Pareto front deviates from the $TH_{sunrise}$ = 0h in the sense that it shows a greater potential for the absolute mitigation of $\overline{eATR20}_{contrail}$ along the complete Pareto front. In figure 4.6, the results for $TH_{sunrise}$ = 6h were more similar to the results for $TH_{sunrise}$ = 0h than the $TH_{sunrise}$ = 3h results. Accordingly, the daytime front for $TH_{sunrise}$ = 6h in figure 4.7 is also closer to that of $TH_{sunrise}$ = 0h than the $TH_{sunrise}$ = 3h front. Moreover, the variability of the daytime Pareto front is mostly horizontal, indicating that the sensitivity of the achievable mitigation of contrail ATR20 to the selected $TH_{sunrise}$ = 0h is higher than the sensitivity of the increase in SOC. This is expected, as the change in $TH_{sunrise}$ affects the contrail aCCF directly, while optimised trajectories and the corresponding $\Delta\overline{eSOC}$ are only affected in a secondary manner.

For daytime flight segments within 3 hours before sunrise, the positions of the winter daytime Pareto fronts lead to the believe that warming contrails are easily avoided, or that deliberately forming extra cooling contrails is achieved at low extra cost. The positive effect for $TH_{sunrise}$ = 3h is diminished for the $TH_{sunrise}$ = 6h results. Hence, for flight segments between 6 and 3 hours before sunrise, warming contrails are harder to avoid and/or airspace supporting cooling contrails is difficult to find.

The nighttime winter Pareto front faces a smaller variation than the daytime winter Pareto front. Again, the deviation for $TH_{sunrise}$ = 3h is in line with the $TH_{sunrise}$ = 3h bar in figure 4.6. The lower $\overline{ATR20}_{contrail}$ of the bar results in a lower absolute mitigation potential. However, while the $TH_{sunrise}$ = 6h bar for winter nighttime in figure 4.6 shows a value close to the $TH_{sunrise}$ = 0h bar, the winter nighttime $TH_{sunrise}$ = 6h Pareto front deviates further from the $TH_{sunrise}$ = 0h Pareto

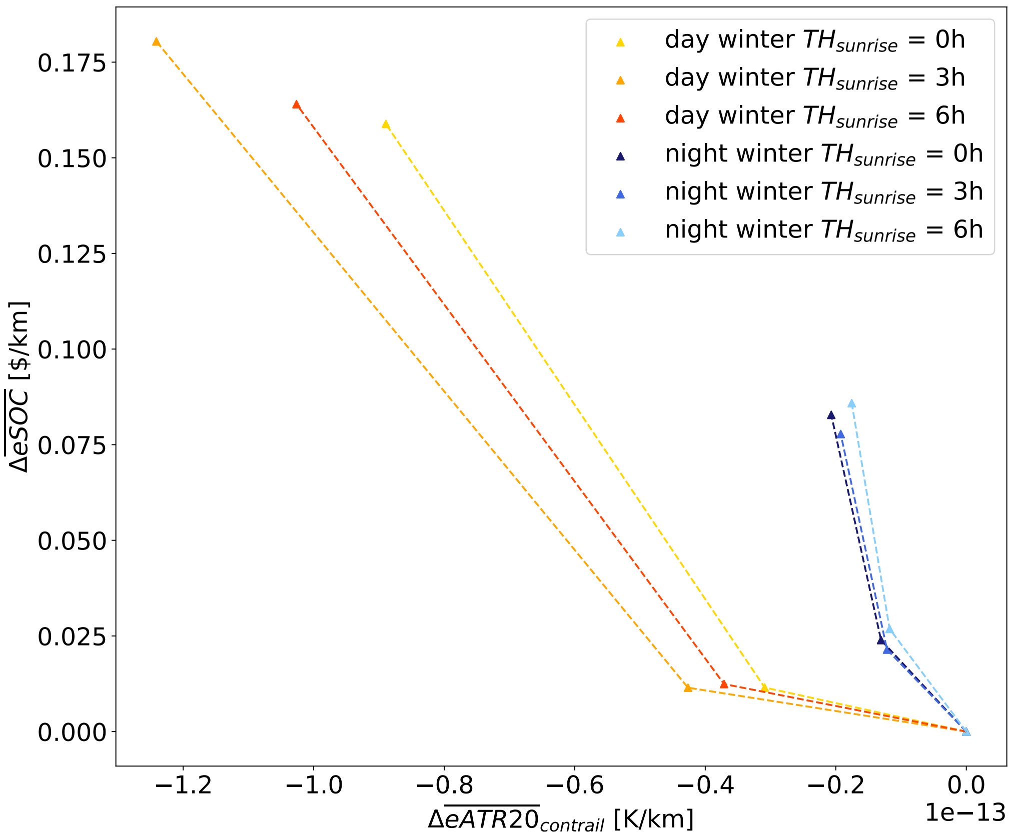

Figure 4.7: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter, generated with waypoint data, $TH_{sunrise}$ = 0, 3 and 6h.

front than the $TH_{sunrise}$ = 3h Pareto front does. The fact that the cost optimal $\overline{ATR20}_{contrail}$ for $TH_{sunrise}$ = 6h is higher than $\overline{ATR20}_{contrail}$ for $TH_{sunrise}$ = 3h does apparently not compensate for the situation faced by mitigation efforts: it seems that in winter nighttime, the avoidance of warming contrails is relatively easy for segments within 3 hours to sunrise. This also holds for nighttime segments between 6 and 3 hours before sunrise.

Then, figure 4.8 shows how the summer waypoint daytime and nighttime Pareto fronts vary with $TH_{sunrise}$. For daytime mitigation, the figure shows that moving from $TH_{sunrise}$ = 0h to $TH_{sunrise}$ = 3h results in a slight loss in maximum mitigation potential, which is restored and even slightly improved for $TH_{sunrise}$ = 6h. This right-and-then-left behaviour with increasing $TH_{sunrise}$ can again be retraced to the behaviour seen for summer daytime in figure 4.6. Overall, the Pareto front for $TH_{sunrise}$ = 3h does not differ largely from the $TH_{sunrise}$ = 0h front, indicating that summer daytime flight segments within 3 hours before sunrise are able to avoid warming contrails or find cooling contrails against approximately the same cost as the daytime segments after sunrise. The more optimal orientation of the $TH_{sunrise}$ = 6h front, tells us that summer daytime flight segments use a relatively low amount of extra cost to avoid warming contrails or find cooling contrails if they are between 6 and 3 hours before sunrise.

According to the bar plot, we expect a left-and-then-right behaviour for the summer nighttime Pareto front. Except for the contrail optimal point, the first (left) does not occur, but the figure does comply with the latter (back right) effect: we find both the $TH_{sunrise}$ = 3h and $TH_{sunrise}$ = 6h front to the right of the $TH_{sunrise}$ = 0h front. This leads to the believe that, for summer nighttime segments within 3 hours until sunrise, the avoidance of warming contrail comes at low extra $\overline{eSOC}$. While the $TH_{sunrise}$ = 3h bar plot was relatively high compared to

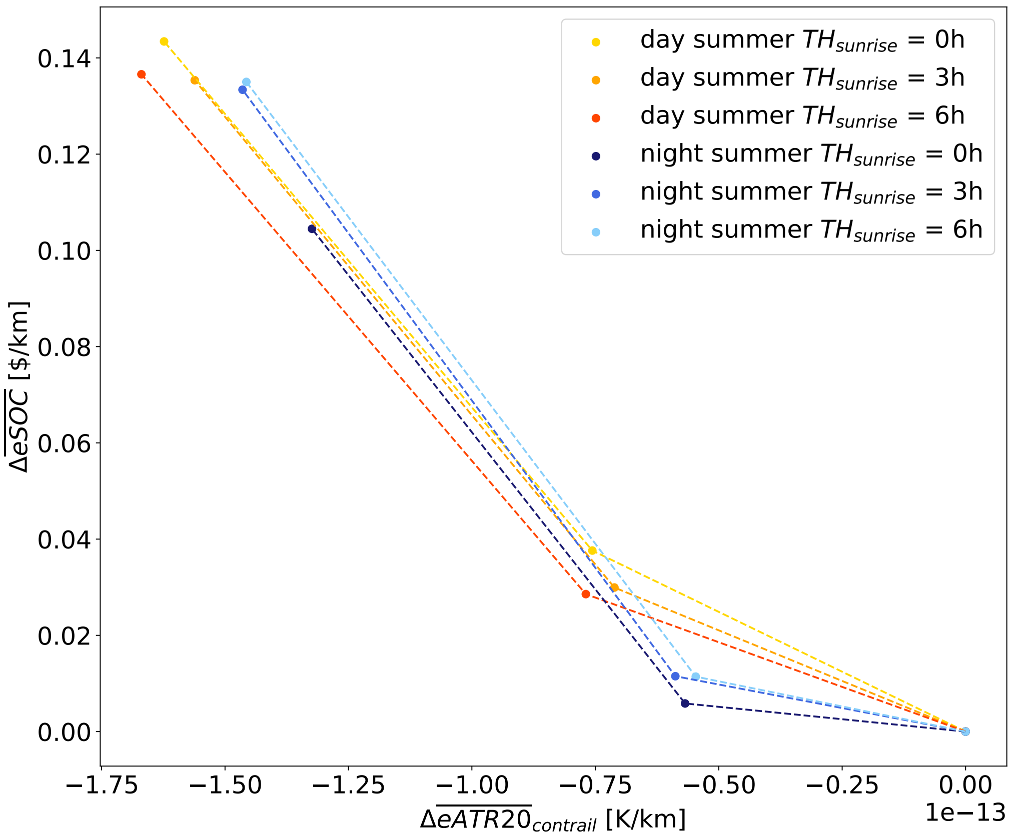

Figure 4.8: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in summer generated with waypoint data, $TH_{sunrise}$ = 0, 3 and 6h.

the $TH_{sunrise}$ = 6h bar, there is a high similarity between the $TH_{sunrise}$ = 3h and $TH_{sunrise}$ = 6h Pareto front. This points to the conclusion that for summer nighttime segments between 6 and 3 hours before sunrise mitigation of $\overline{eATR20}_{contrail}$ is relatively costly. In general the Pareto fronts of figure 4.7 and figure 4.8 show some variability with $TH_{sunrise}$, but not enough result in diverse conclusions. Hence, mitigation results for waypoint data are judged to be robust with respect to a changing $TH_{sunrise}$. However, one should be careful when comparisons are made between the summer daytime and nighttime front. Changes in $TH_{sunrise}$ can result in a switch of relative position. Thus, while nighttime mitigation may be more cost-effective for one $TH_{sunrise}$, this may not be the case for another $TH_{sunrise}$.

For winter and summer the variation of $TH_{sunrise}$ is also performed for flight data and the resulting figures were analysed. The flight data figure for the winter Pareto fronts was largely in agreement with figure 4.7, hence, this figure is omitted in this section and included in the appendix (see figure A.1). As in particular the daytime Pareto front of the winter flight data and winter waypoint data were corresponding to each other, we adopt the conclusions drawn for winter daytime flight segments also for winter daytime flights: for daytime flights within 3 hours before sunrise, warming contrails are easily avoided, or that air space supporting cooling contrails is easily found. For flights between 6 and 3 hours before sunrise, warming contrails are harder to avoid or air space supporting cooling contrails is difficult to find. For winter nighttime, the flight data showed hardly any variability with $TH_{sunrise}$. For nighttime flights the conclusions for waypoint data are thus not copied. The lack of variability for winter nighttime flight indicates that nighttime flights within 6 hours before sunrise spend an amount of SOC on the mitigation of contrail ATR20 that is comparable to the amount spent for the rest of the nighttime flights. Overall, winter results for flight data are thus robust, especially for winter nighttime flights.

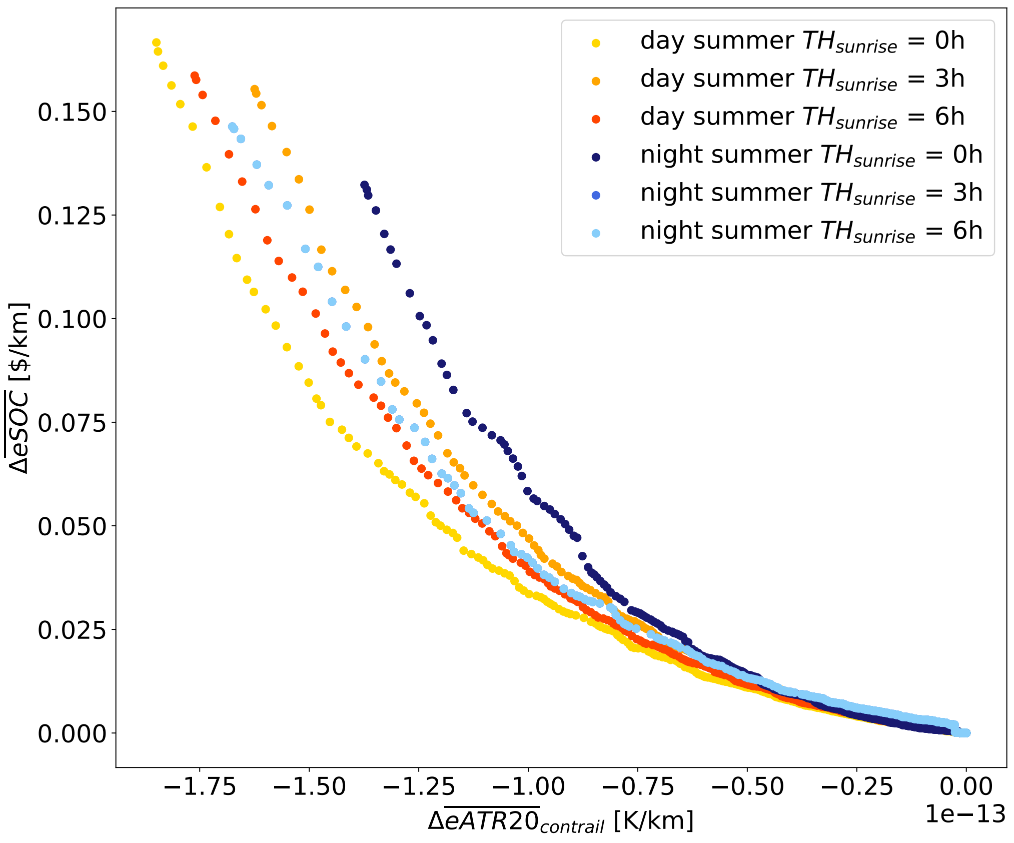

Figure 4.9: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in summer generated with flight data, $TH_{sunrise}$ = 0, 3 and 6h.

Then, figure 4.9 shows how the summer flight data daytime and nighttime Pareto fronts vary with $TH_{sunrise}$. Here, we observe some differences with respect to figure 4.8. For daytime mitigation, figure 4.9 shows that moving from $TH_{sunrise}$ = 0h to $TH_{sunrise}$ = 3h again results in a loss in maximum mitigation potential, which is somewhat (but not completely) restored for $TH_{sunrise}$ = 6h. More than we have seen for $TH_{sunrise}$ = 3h in figure 4.8, the complete Pareto front in figure 4.9 moves to the right of the original front of $TH_{sunrise}$ = 0h. Different from the waypoint data, we can conclude that summer daytime flights within 3 hours before sunrise avoid warming contrails or find cooling contrails against relatively high cost. The more optimal orientation of the $TH_{sunrise}$ = 6h front, tells us that summer daytime flights use a relatively low amount of extra cost to avoid warming contrails or find cooling contrails relatively easy if they are between 6 and 3 hours before sunrise.

For the nighttime summer Pareto fronts we find the $TH_{sunrise}$ = 6h front to the left of the $TH_{sunrise}$ = 0h front, with the $TH_{sunrise}$ = 3h front exactly positioned behind it. The exact overlap of those two fronts is unlikely and probably the results of a processing error that could not be identified before publishing this work. In case the overlap in results is correct anyhow, then this leads to the believe that for summer nighttime flights within 3 hours before sunrise mitigation of $\overline{eATR20}_{contrail}$ is relatively costly. According to the bar plot, the nighttime front for $TH_{sunrise}$ = 6h should be closer to the $TH_{sunrise}$ = 0h front than shown by figure 4.9. It is thus concluded that mitigation efforts for summer nighttime flights between 6 and 3 hours before sunrise are relatively costly as well. In case the overlap of results is incorrect, the same conclusions can be drawn, although they only hold for the 6 hours until sunrise overall and not necessarily for both 0-3 hours before sunrise and 3-6 before sunrise specifically. The shape and values of the flight data Pareto fronts seem to be robust with a varying $TH_{sunrise}$. However, as seen for waypoint data, caution is required for a relative comparison between the daytime and nighttime front, as their relative positions may be switched by a change in $TH_{sunrise}$.

4.2.3 Variability of Mitigation of Contrail ATR20 Without Extra Cooling

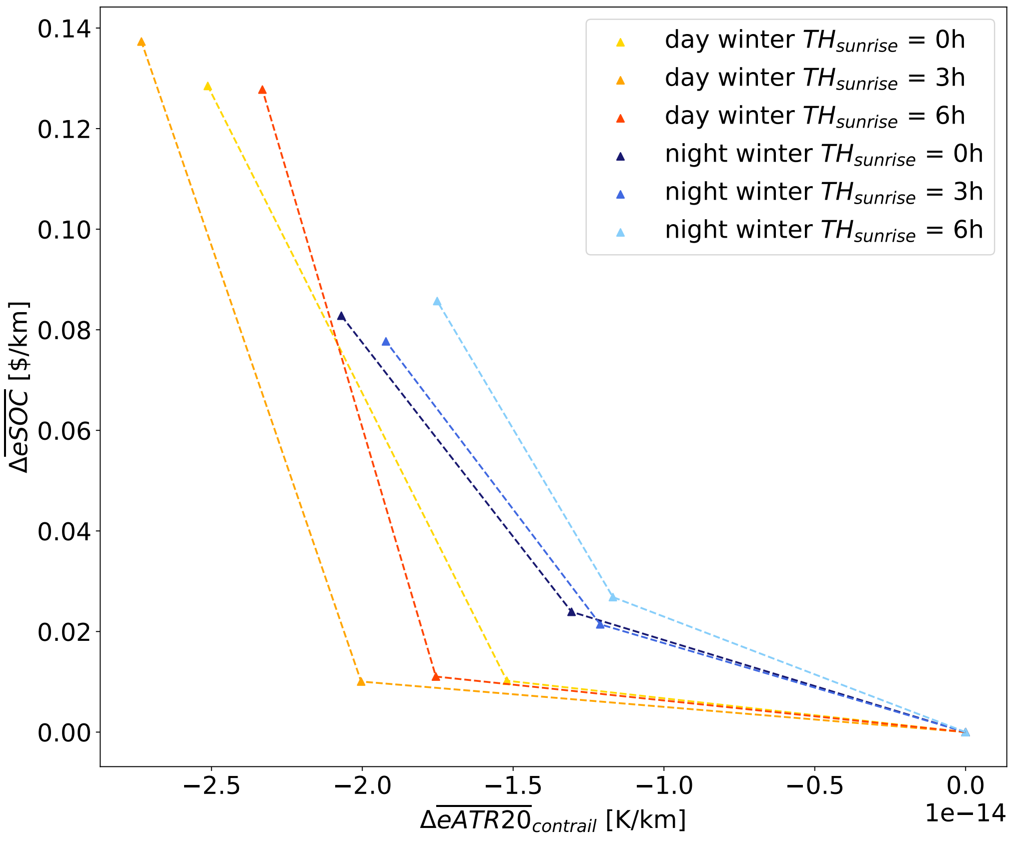

Figure 4.10: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter, generated with waypoint data, $TH_{sunrise}$ = 0, 3 and 6h. Extra cooling is excluded.

This section shows the variability of results for waypoint data with excluded extra cooling. Figure 4.10 shows the variability for winter and figure 4.11 shows the variability for summer. As shown in table 4.1, the exclusion of extra cooling only affects nighttime flight segments through a marginal change in the total number of nighttime flight segments. The effect on the results should be negligible for all three values of $TH_{sunrise}$, which is confirmed by a quick comparison between the nighttime Pareto fronts of figure 4.7 and 4.10 and figure 4.8 and 4.11. Hence, this section focuses solely on the variation of the results without extra cooling for daytime segments.

At first glance, the variation of figure 4.10 with $TH_{sunrise}$ agrees with the variation seen in figure 4.7. However, in figure 4.10, we observe something interesting at the contrail optimal end of the daytime Pareto fronts. Although all three fronts are considerably affected by the replacements of segments that are extra cooling, the maximum mitigation of $\overline{eATR20}_{contrail}$ has decreased more for $TH_{sunrise}$ = 3h and 6h than for $TH_{sunrise}$ = 0h. Therefore, we conclude that daytime flight segments within 6 hours before sunrise are more dependent on extra cooling than the daytime segments after sunrise, and that is mostly holds for the contrail optimal solution. Still, conclusions that can be drawn from figure 4.10 do not significantly vary with $TH_{sunrise}$. Therefore, also without extra cooling allowed, winter mitigation results are found to be robust while facing a change in $TH_{sunrise}$.

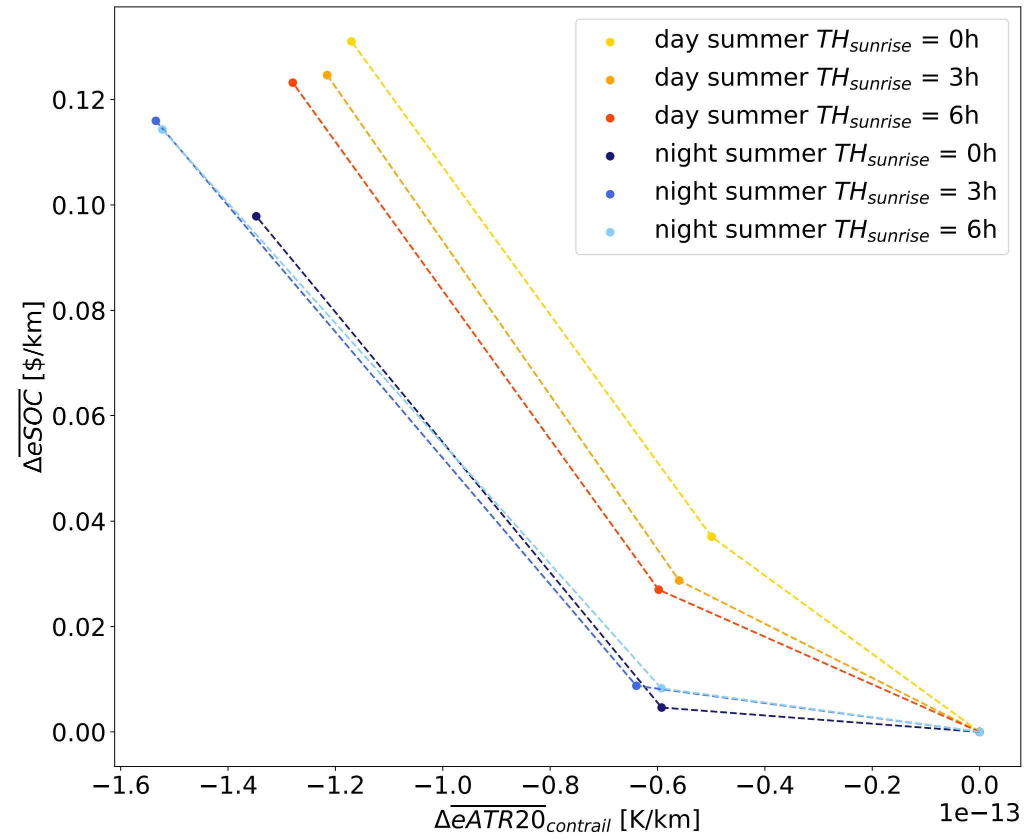

Figure 4.11: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in summer generated with waypoint data, $TH_{sunrise}$ = 0, 3 and 6h. Extra cooling is excluded.

Whereas the winter Pareto fronts of 4.10 showed a change in position and shape with respect to 4.7, for summer the change from 4.8 to 4.11 seems to be about position only. All three Pareto's are affected by the replacements of segments with extra cooling along their whole front, but the rightward shift is most pronounced for $TH_{sunrise}$ = 0. This leads to the conclusion that in general daytime segments within 6 hours before sunrise are less dependent on extra cooling than daytime segments after sunrise. Otherwise, the summer results are not heavily altered by an increasing $TH_{sunrise}$ and are thus robust. As without extra cooling the difference between the summer daytime and nighttime Pareto front is more pronounced than with extra cooling, the robustness of results is even stronger with extra cooling excluded.

4.3 Search for a New Pareto Front

This section presents the results that answer to the third research objective of this thesis. The methods of analysis for these results were described in Section 3.3.4. Using the approach of that section, new theoretical points within the Pareto plot are found, that aim to employ the difference between daytime and nighttime flights to come to better mitigation results. First, figure 4.12 is presented, showing the search for new points in winter. Then, figure 4.13 presents new points found for the summer season.

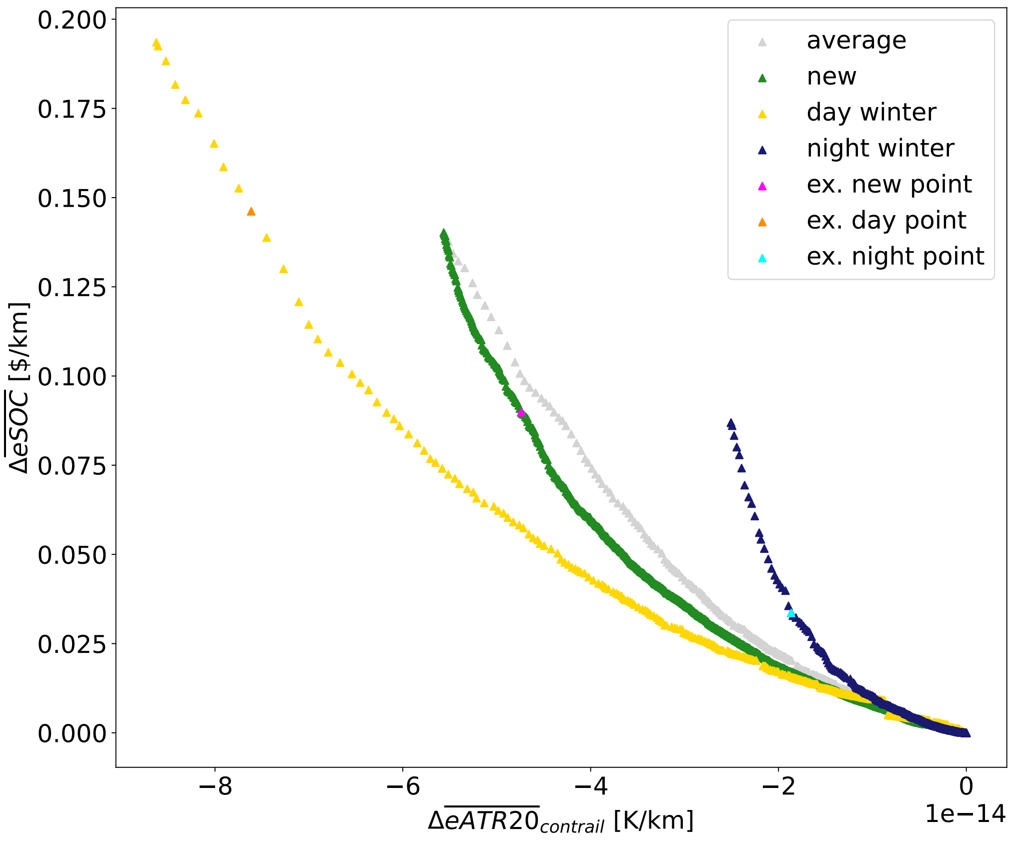

Figure 4.12: New found points outperforming the average of the day and night winter Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ generated with flight data, $TH_{sunrise} = 0\ h$. The points labelled with 'ex.' indicate an illustrative new found point and its source points.

Figure 4.12 shows the newly found points (green) outperforming the average points (grey) of the day and night Pareto front of $\overline{eSOC}$ vs. $\overline{eATR20}_{contrail}$ for winter. As is visible in the figure, the green points perform better than the average points in terms of eco-efficiency. The figure shows that new points around the middle part of the Pareto front feature the largest distance from the average Pareto front, i.e., the gain in eco-efficiency is more pronounced in the middle part than near the tail ends of the Pareto front. The pink point highlights and exemplary newly found point and is the average of the orange highlighted daytime point and the turquoise highlighted nighttime point. The example point shows well how it is beneficial to spend more extra $\overline{eSOC}$ during daytime and less $\overline{eSOC}$ during nighttime to improve the overall achievable $\Delta\overline{eATR20}_{contrail}$.

Figure 4.13 shows the newly found points for the summer Pareto front. Compared to figure 4.12, the green line of new points is now much closer to the average, represented by the grey dots. Contrary to figure 4.12, the newly found points in 4.13 cannot benefit from the large distance between the daytime and nighttime Pareto front. Nonetheless, it is in favour of the newly found points for the summer that the shape of the nighttime summer Pareto front is somewhat rugged. A faint imprint of the rugged pattern of the nighttime fronts is observable in the line of newly found points. The new points that managed to reach the most distance from the grey dots, are clearly those that make use of the bulges of the nighttime front. Overall, it seems that the winter is more fruitful if one desires to employ the difference between

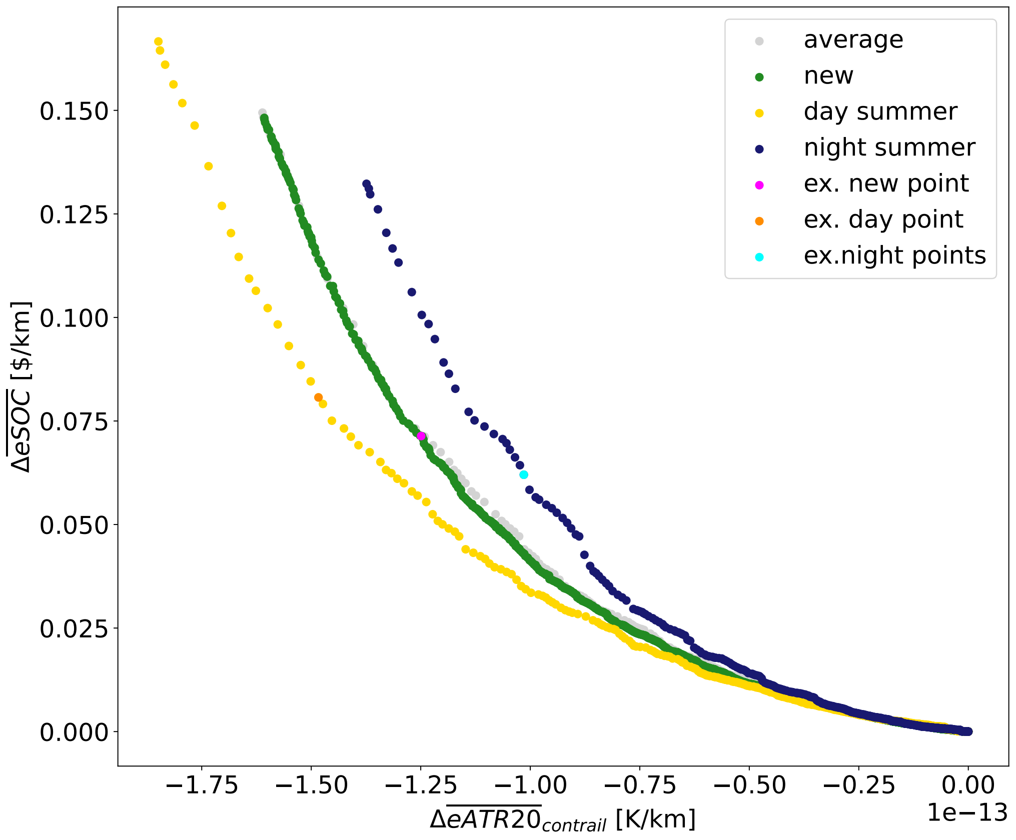

Figure 4.13: New found points outperforming the average of the day and night summer Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ generated with flight data, $TH_{sunrise}$ = 0 h. The points labelled with 'ex.' indicate an illustrative new found point and its source points.

daytime and nighttime for the mitigation of contrail climate impact. Both figure 4.12 and 4.13 confirm the theoretical possibility to make use of the difference between daytime mitigation and nighttime mitigation to enhance the overall mitigation potential. Additionally, the figures show that this enhancement is often reached by spending more extra SOC during daytime than during nighttime.

Chapter 5

Verification and Validation

To check if the thesis method has been carried out correctly and whether the results correspond to reality, this chapter presents the verification and validation of this work. Verification is discussed in Section 5.1 and validation is discussed in Section 5.2.

5.1 Verification

This section presents several steps that are taken to verify the correctness of calculations. The verification steps should confirm that no mistakes are made in the set up of the methodology and the execution of this methodology.

The first verification approach consists of unit tests performed on the code written for this thesis. For this purpose the complete algorithm is divided in small chunks (units) that are responsible for a simple task. Whether a unit performs its task as expected is verified by using realistic input data and additionally boundary cases (e.g., a zero or infinity input). If any error was raised, the code was modified until the error was corrected. After running each unit of code, the output data was checked for anomalies. When necessary, the code was further improved until no data anomalies occurred. After unit-testing the small code chunks, larger chunks were tested in a similar manner, verifying that the smaller chunks of code interact correctly. Lastly, the complete algorithm was tested following the same method. An example of such an anomaly check is performed on the nighttime Pareto front for the results without extra cooling. By definition these result should barely differ from the results where extra cooling is allowed, since nighttime segments cannot feature a cooling contrail. The similarity between nighttime results with and without extra cooling was indeed confirmed.

The second verification method focuses on the used selection of flights. As the number of selected routes and flight instances is limited, it is questionable whether the selection is representative for all European winter and summer flights. To verify whether the flight selection is sufficiently representative, the results for waypoint data will be recreated for a wider selection of flights. The verification flight plan corresponds to the ATM4E European flight plan comprising the top 100 European routes in terms of available seat kilometers in 2018. The flights depart at midnight and at noon and span all days of December (2017), January and February (2018) for winter and all days of June, July and August (2018) for summer. The simulation is only performed for $TH_{sunrise}$ = 0h. Further settings correspond exactly to the description in Chapter 3.

Figure 5.1 shows the bar plot results for the verification flight plan. At first sight, the plot corresponds well to the results of figure 4.1. For nighttime, the found specific contrail ATR20 almost exactly agrees between the figures. For daytime, the verification flight plans shows a larger value for $\overline{ATR20}_{contrail}$ than the thesis flight plan. Although the mean daytime values of $\overline{ATR20}_{contrail}$ are well within the standard deviation of figure 4.1, this means that the thesis flight plan leads to a slight underestimation of the cost optimal daytime $\overline{ATR20}_{contrail}$. Moreover, figure 5.1 shows that cost optimal daytime and nighttime $\overline{ATR20}_{contrail}$ are not as equal as observed for the thesis flight plan. Nonetheless, their magnitudes remain comparable. Lastly, figure 5.1 confirms the difference between winter and summer, where summer features a specific contrail ATR20 that is a few times the magnitude of that of winter.

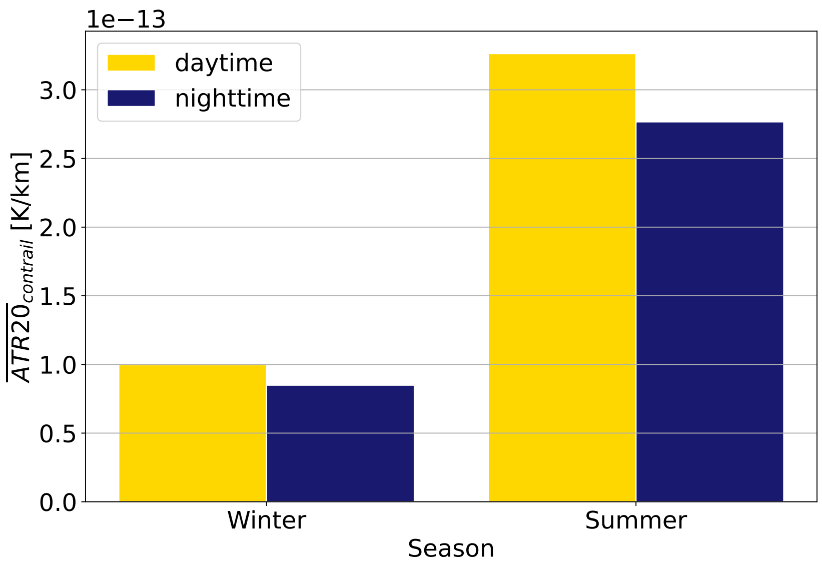

Figure 5.1: Specific, SOC optimal day and night contrail ATR20 for winter and summer, $TH_{sunrise}$ = 0 *h*. The data originate from the ATM4E European flight plan.

Next, figure 5.2 shows the waypoint Pareto results for the verification flight plan. With respect to figure 4.3, figure 5.2 shows discrepancies in a number of aspects: what stands out first is that in figure 5.2 the summer daytime front is found to the lower left of the summer nighttime front and is thus more eco-efficient. In figure 4.3 the summer daytime and nighttime front where semi-overlapping, where the daytime front was found slightly to the upper right of the nighttime front. A closer look at the points in figure 4.3 and 5.2 shows that the thesis flight plan underestimates the summer daytime eco-efficiency and overestimates the summer nighttime eco-efficiency. To verify whether the summer daytime front is actually and robustly more eco-efficient than the summer nighttime front, more experiments are recommended. Nonetheless, the summer eco-efficiency of figure 5.2 falls within one standard deviation of figure 4.3. The winter eco-efficiency matches fairly well between the figures. Another notable difference is found for the contrail optimal solutions. Both in absolute and relative sense, the maximum reduction in $\overline{eATR20}_{contrail}$ is significantly larger in figure 5.2 than we have seen in figure 4.3. This holds for both daytime and nighttime and both for winter and summer. The discrepancy indicates that the thesis flight plan runs out of mitigation options earlier than the ATM4E flight plan. The ATM4E flight plan has a larger maximum mitigation potential, which also comes at a higher amount of extra SOC. Additionally, a comparison can be made between figure 5.2 and figure 4.4. Although the two plots convey information of different data types (waypoint data and flight data, respectively), they show high similarity in shape and therefore lead to similar conclusions.

Then, figure 5.3 shows the waypoint Pareto results with removed extra cooling for the verification flight plan. By comparing figure 4.5 to figure 5.3, the effect of removed extra cooling seems to be verified. Both daytime Pareto fronts are significantly affected, where the resulting change is largest for the winter daytime front. A difference between the thesis results and these verification results is that the summer daytime front is not affected to such an extent that it shifts to the right of the summer nighttime front. Again, the verification results show that the thesis flight plan underestimates the summer daytime eco-efficiency and overestimates the summer nighttime eco-efficiency. Yet, the verification summer eco-efficiency is within a standard deviation of the results for the thesis flight plan. The verification results for winter eco-efficiency again agree well with previous results. Just as for figure 5.2, we see in figure 5.3 that the ATM4E flight plan results in larger values for $\Delta\overline{eSOC}$ and $\Delta\overline{eATR20}_{contrail}$ for

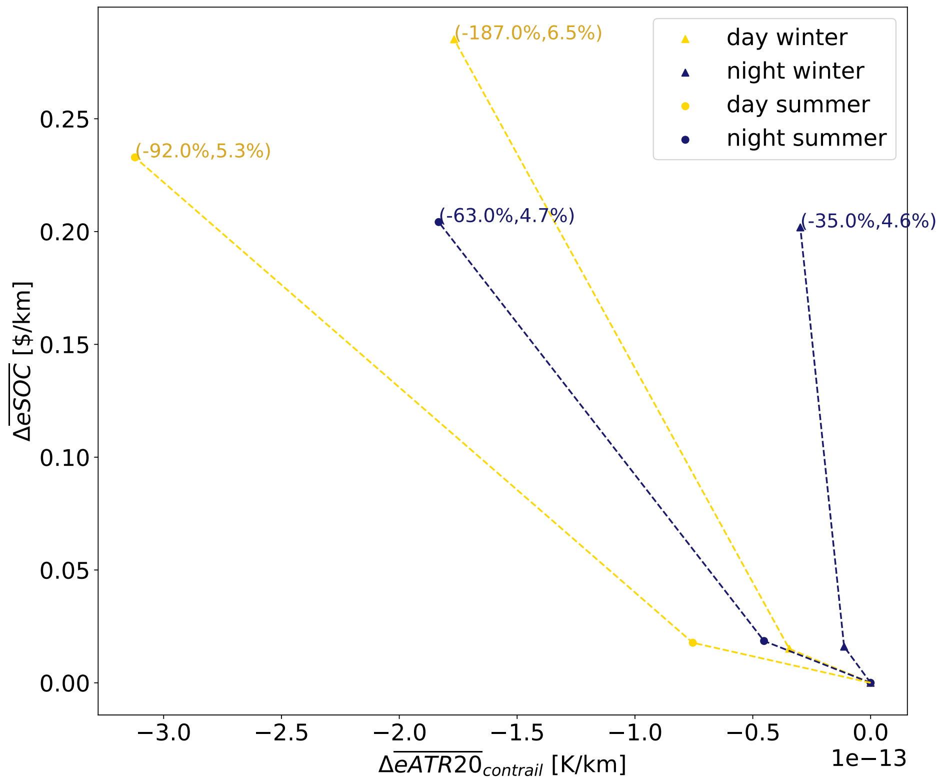

Figure 5.2: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter and summer generated with waypoint data, $TH_{sunrise} = 0\ h$. The percentages placed at the contrail optimal solution indicate the relative differences to the SOC optimal solution. The data originate from the ATM4E European flight plan.

the contrail optimal solution. This again confirms that he ATM4E flight plan has a larger maximum mitigation potential, paired with a higher amount of extra cost. More than in figure 4.5, the removal of extra cooling from the verification simulation seems to bring daytime and nighttime results closer to each other. Most remarkable is that the contrail optimal solutions all feature a similar $\Delta\overline{eATR20}_{contrail}$, whereas figure 4.5 showed that the contrail optimal daytime points featured a $\Delta\overline{eSOC}$ that was considerably higher than for the contrail optimal nighttime points. The nearly equal values for $\Delta\overline{eSOC}$ are also observed for the intermediate solutions. Regarding the cost of the contrail optimal solution, the verification results show that the thesis flight plan leads to an overestimation of the difference between daytime mitigation and nighttime mitigation without extra cooling contrails.

The fact that we analyse average Pareto fronts both for waypoint data and flight data presents another opportunity for verification. In Chapter 4, we made use of a key difference between waypoint data and flight data. Namely that the first can distinguish day and night at segment level, while the latter can only make this distinction for a complete flight. However, the postprocessing of waypoint data allows us to shift the day-night distinction on segment level to a flight level distinction. If, after this shift, the three-point Pareto fronts from the waypoint data correspond to the Pareto fronts from the flight data, we can verify that the calculations for both data types are performed correct. As explained with figure 3.11, the contrail optimal and SOC optimal point of the flight data Pareto front are well represented, but intermediate points are prone to a bias towards lower eco-efficiency. Therefore we still expect a small discrepancy between the $\sim$0.5% extra SOC solution, while the end points are expected to match exactly.

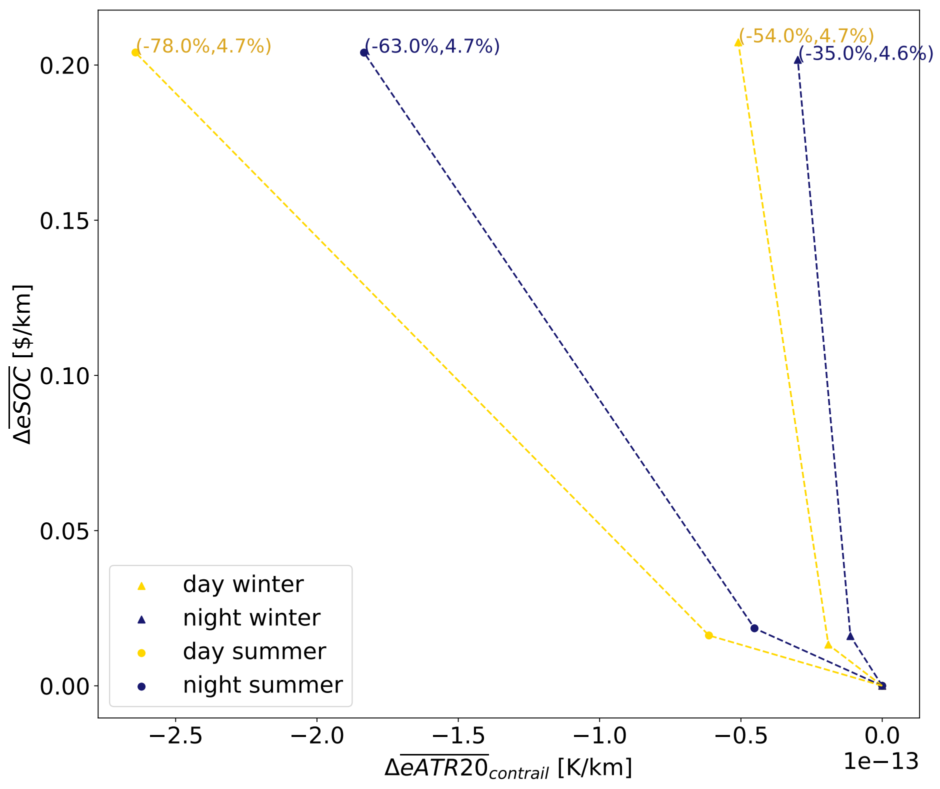

Figure 5.3: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter and summer generated with waypoint data, $TH_{sunrise}$ = 0 h. Extra cooling is excluded. The percentages placed at the end-points of the fronts indicate the change w.r.t. the SOC optimal solution. The data originate from the ATM4E European flight plan.

The result of this verification step is presented in figure 5.4. The figure shows a general agreement between the waypoint data and the flight data results. Since the cost optimal solution is used as reference point for plotting, the coinciding of the waypoint and flight data is automatically ensured at the cost optimal end of the Pareto fronts. However, the ∼0.5% extra SOC solution and the contrail optimal solution of waypoint and flight data do not exactly correspond. This is mainly unexpected for the contrail optimal solution, given the explanation in figure 3.11. The contrail optimal waypoint solution is consistently found to the lower right of the flight data contrail optimal point. This leads to the believe that the discrepancy is the result of a difference in the calculated flight distances, which is used for the normalisation of the results. For this normalisation, the different approaches were described in Chapter 3. A test confirmed that the method associated with waypoint data estimates a larger overall flight distance than the method for flight data. This difference results in a smaller estimate of $\Delta\overline{eSOC}$ and $\Delta\overline{eATR20}_{contrail}$ for waypoint data. Given that the source of discrepancy is know, figure 5.4 gives confidence in the correct execution of the methodology.

This verification section described how the code for this thesis was tested for correct results. Furthermore, this section showed whether the limited air traffic sample led to representative results for a larger amount of European air traffic. The cost optimal results turned out to very representative and also the eco-efficiency of most Pareto fronts matched well with a larger air traffic sample. However, the contrail optimal point of the main thesis results shows a maximum mitigation potential that is too pessimistic. Without extra cooling, the contrail optimal solution for daytime and nighttime is more similar than the main thesis results present. Lastly, the processing algorithms for waypoint data and for flight data were

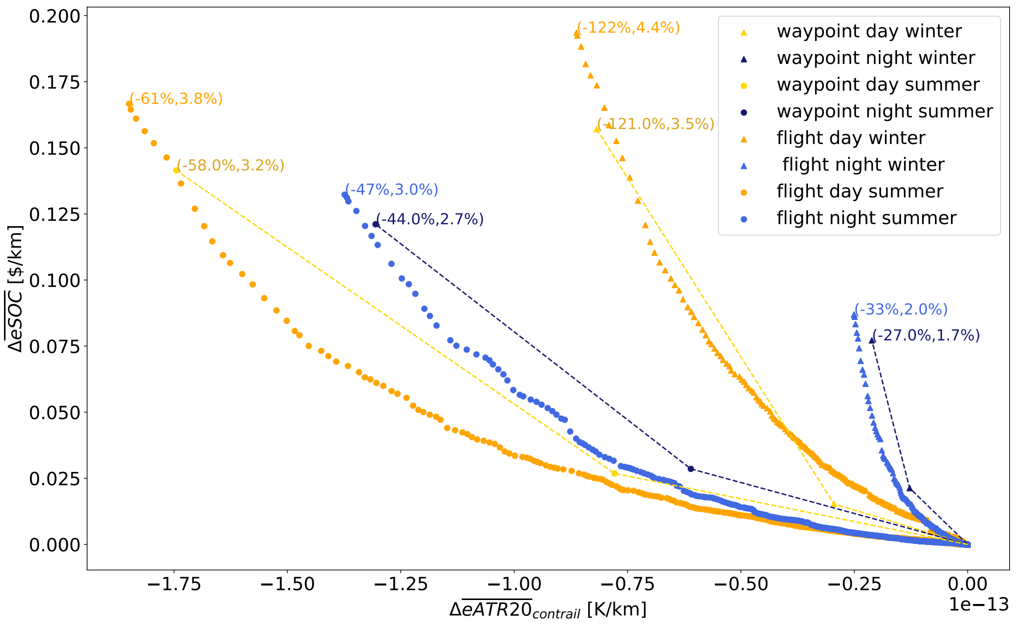

Figure 5.4: Pareto front of $\Delta\overline{eSOC}$ vs. $\Delta\overline{eATR20}_{contrail}$ for day and night in winter and summer, generated with waypoint data and flight data, $TH_{sunrise}$ = 0. For the purpose of verification the waypoint data Pareto front mimics the distinction between day and night for flight data.

verified against each other.

5.2 Validation

The options for the validation of the thesis results are limited. Real world measurements of contrail climate impact are scarce, and measurements of mitigation potential for contrail climate impact are even scarcer. Hence, to validate the outcomes this thesis, its results are compared to the results of earlier publications. The results of these publications should not conflict with the results of this thesis, unless the discrepancies can be explained. This section will discuss how the thesis results compare to a selection of relevant publications and shall assess the validity of this work.

The first work that is used to validate the thesis results is that of Yin et al. (2018). Yin et al. (2018) deals with the avoidance of contrail formation as a trade-off against flight time and also makes use of AirTraf as air traffic optimisation tool. Since the avoidance of contrails is the only way to mitigate contrail climate impact during nighttime, and since SOC is partly determined by flight time, the nighttime results of this thesis should be comparable to the results of Yin et al. (2018). While comparing figure 10 and 11 from Yin et al. (2018) to the nighttime results in figure 4.3, we see that the shape of the Pareto fronts are matching. Moreover, the summer relative change in contrail distance of Yin et al. (2018) seems to agree reasonably well with the summer relative change in contrail ATR20. However, the relative change in flight time does not correspond with the relative change in costs. A large difference is found for the winter nighttime mitigation. According to Yin et al. (2018), winter should (relatively) be more eco-efficient than summer, which is not the case for the thesis data. The data used for verification (figure 5.2) shows a slightly better agreement with the results in Yin et al. (2018), especially for summer nighttime and to a lesser extent for winter nighttime too. Still the winter discrepancy remains large. Possible sources for the discrepancies are the difference in region (North Atlantic), year (2011), aircraft model (Airbus A330) and the difference between time and SOC as optimisation objective.

Next, results from Yamashita et al. (2020) table 4 are compared with our results. The results of Yamashita et al. (2020) are generated for a one day winter simulation of 130 fights across the Atlantic ocean. Among others, Yamashita et al. (2020) provides values for the SOC optimal, minimum contrail formation and climate optimal solution (minimum total ATR20). Since the latter is very much driven by contrail ATR20 (e.g., see Lührs et al. (2020)), the climate optimal solution is comparable tot the contrail optimal solution. We again compare figure 4.3 and see that the relative winter nighttime $\Delta\overline{eATR20}_{contrail}$ approximately agrees with the relative change in total ATR20 for the minimum contrail formation solution. The relative $\Delta\overline{eSOC}$ in figure 4.3 is much smaller than for the results of Yamashita et al. (2020). The climate optimal solution of Yamashita et al. (2020) probably also involves cooling contrails, therefore the comparison involves both the nighttime and daytime front of figure 4.3. In terms of relative $\Delta\overline{eATR20}_{contrail}$ the value of Yamashita et al. (2020) indeed lies between the winter daytime front and the nighttime front of figure 4.3, but the relative $\Delta\overline{eSOC}$ differs significantly. Again the verification data seems to agree slightly better than the data obtained from the thesis flight plan. Discrepancies can be attributed to the total ATR20 as optimisation objective and again to differences in location (North Atlantic), time (December 2015) and aircraft model (Airbus A330).

Lastly, the results are compared to those of Lührs et al. (2021), who simulated over 13000 daytime flights for a 2015 December day over Europe. The flights are optimised for total ATR20, which is heavily dominated by the mitigation of contrail ATR20. The comparison focuses on figure 7b of Lührs et al. (2021) and the winter daytime Pareto front of figure 4.4. In its shape, the Pareto front of Lührs et al. (2021) matches well with the winter daytime Pareto front of figure 4.4. However, some significant differences are noted for the climate (Lührs et al. (2021)) or contrail (this work) optimal point: while our results show over 100% reduction in $\overline{eATR20}_{contrail}$, the results of Lührs et al. (2021) 'only' achieve a reduction of less than 80% of the overall ATR20. Moreover, the costs that correspond to these maxima are of different magnitude. Our data show a maximum of 4.4% extra $\overline{eSOC}$, while the extra time and fuel reaches around 13% for the results of Lührs et al. (2021). Again there is a set of reasons that can cause the discrepancies between the results. The results of Lührs et al. (2021) are generated for a single day with high contrail formation and was optimised with minimum total ATR20 as objective.

In summary, the validation of the results of this thesis is difficult due to a lack of measurement data and comparable literature. The literature that is available, often deals with a test case that differs from ours in several important aspects. Overall, values for nighttime $\Delta\overline{eATR20}_{contrail}$ are validated reasonably well. Future work is needed to validate daytime results and $\Delta\overline{eSOC}$ values for both daytime and nighttime.

Chapter 6

Discussion

This chapter discusses the results shown in Chapter 4. Section 6.1, 6.2 and 6.3 will address the first, second and third research objective of the thesis, respectively. Lastly, section 6.4 discusses the assumptions and uncertainties of which the reader should be aware to properly value the conclusions of this work.

6.1 Comparing Day and Night Contrail Climate Impact of Optimised Trajectories

This section will discuss how the results of Chapter 4 are tied to the first objective of the thesis: to analyse the daytime and nighttime difference in contrail ATR20 for a set of eco-efficient trajectories. For this purpose, this section will build upon the observations of Section 4.1. First, the results for winter will be discussed, then for summer. Lastly, interesting differences between these two seasons will be highlighted.

Winter

For the winter season, we have seen in Section 4.1.1 that the difference between day and night specific contrail climate impact is negligible for the cost optimal solution. For solutions that are more contrail optimal, we have seen in Section 4.1.2 that daytime mitigation achieves results that are significantly more eco-efficient than nighttime results. This holds if optimisation is analysed at segment level, but also if data is assessed at the level of a complete flight. Winter daytime mitigation of contrail climate impact is not only more eco-efficient than winter nighttime mitigation, daytime mitigation also carries further than nighttime mitigation: maximum daytime mitigation achieves a multitude of the magnitude of the nighttime mitigation, although the daytime maximum also comes at a much larger cost. Section 4.1.3 indicated that the magnitude of winter daytime mitigation is largely depended on the formation of extra cooling contrails. Nonetheless, the eco-efficiency of daytime mitigation remains slightly greater than the eco-efficiency of nighttime mitigation, even if extra cooling is excluded from the mitigation efforts.

The comparability of cost optimal daytime and nighttime contrail ATR20 is unexpected: while nighttime contrails have a warming effect due to their interaction with LW radiation, daytime contrails have this same warming effect and a cooling effect due to their SW interaction. On average, one thus may expect that the warming effect is (partly) offset by a cooling effect during daytime, resulting in a lower specific contrail climate impact for day than for night (Stuber and Forster (2007), Newinger and Burkhardt (2012)). This unexpected result may be caused by an inequality in *potcov*, where daytime features a larger *potcov* than nighttime. This inequality is indeed confirmed by an inspection of the meta data, but not supported by findings of, e.g., Newinger and Burkhardt (2012).

The result that daytime mitigation is more eco-efficient than nighttime is expected. Besides the option to avoid the formation of warming contrails, a daytime flight section has an extra mitigation option compared to a nighttime segment: the formation of cooling contrails can be employed. The same line of thought is an explanation for the higher maximum daytime mitigation. Because daytime mitigation can make use of the formation of extra cooling contrails, it lakes longer (that is, takes more extra SOC) until a trajectory optimisation runs out of further mitigating options.

The reliance of winter mitigation on extra cooling contrails is expected as well: earlier work shows that the PCC for the Northern Hemisphere is relatively large in winter (Fichter et al. (2005), Yin et al. (2018)). Even though the SOC optimal solutions did not confirm this image, more contrail optimal solutions may correspond to the results of, e.g., Yin et al. (2018). Therefore, a change in trajectory to mitigate net contrail climate impact in winter can thus relatively easy resort to the formation of an extra cooling contrail. This is what indeed seems to happen for the winter results.

These winter results tell us that the contrail contributions of aviation's winter climate impact can be mitigated most effectively during daytime. At a similar cost, daytime mitigation achieves a larger reduction of contrail ATR20. When a policy aims to avoid geoengineering while mitigating contrail ATR20, care should be taken during daytime. The results suggest that extra cooling contrails are easily created in winter, implying that extra cooling contrails are also easily formed unintentionally.

Summer

For the summer season, we have seen in Section 4.1.1 that the difference between day and night specific contrail climate impact is negligible for the cost optimal solution. Considering waypoint data, we have seen in Section 4.1.2 that low cost mitigation might be more eco-efficient during nighttime, while the difference in eco-efficiency is negligible in the higher cost region. A reverse picture is painted by flight data: low cost mitigation features only a small difference between daytime and nighttime. At higher extra cost, the difference between daytime and nighttime becomes more pronounced, where daytime mitigation shows more eco-efficiency than nighttime mitigation. Furthermore, both waypoint and flight data agree that summer daytime mitigation carries over 10% further than nighttime mitigation, even though the daytime maximum also comes at a much larger cost. Lastly, Section 4.1.3 indicated that the magnitude of summer daytime mitigation is significantly depended on the formation of extra cooling contrails. When extra cooling is excluded from the mitigation efforts, the eco-efficiency of nighttime mitigation outperforms the eco-efficiency of daytime mitigation.

As described for winter, the similar magnitude of cost optimal specific contrail climate impact contradicts the expectation that daytime features a lower net contrail climate impact than nighttime. The possible presence of cooling contrails does not result in a significantly lower daytime specific contrail climate impact.

That the difference between daytime and nighttime is less outspoken for summer does not come as a surprise. The Northern hemisphere summer features a relatively low PCC in summer (Fichter et al. (2005), Yin et al. (2018)), resulting in fewer options to from extra contrails that are cooling. Likely, daytime mitigation can thus exploit the cooling advantage to a lesser extent.

That daytime mitigation again carries further than nighttime mitigation is not surprising. Although we have seen that, at medium costs, the daytime mitigation only differs slightly from nighttime mitigation, we do see that at high cost daytime mitigation does profit from the ability to create extra cooling contrails.

The summer results provides the reader with an important message. If the mitigation of contrail climate impact can rely on extra cooling contrails, daytime mitigation is more effective than nighttime mitigation. Without extra cooling contrails, nighttime mitigation is more eco-efficient than daytime mitigation.

Seasonal Differences

While comparing the results for winter and summer, we see some noteworthy differences between the seasons. The cost optimal magnitude of summer specific contrail ATR20 is significantly larger than in winter. In absolute sense, this results in a far greater mitigation potential during summer than during winter. This holds for waypoint data, flight data, and waypoint data without extra cooling. Relatively, the maximum nighttime mitigation potential of summer tops that of winter and does so consistently for all three data types. For waypoint data and flight data, the relative daytime mitigation magnitude in winter outperforms the summer daytime mitigation magnitude. Waypoint data without extra cooling forms the exception to this. This highlights another key difference between the seasons. Winter daytime mitigation is more dependent on the formation of extra cooling contrails than summer daytime mitigation.

The results chapter also showed that the *potcov* for cost optimal flights is larger in summer than in winter. This is remarkable, as studies like Yin et al. (2018) and Fichter et al. (2005) found an opposite seasonal cycle for cruise altitude PCC. As discussed in Section 4.1.1, this discrepancy may be caused because of the high mean altitude for cost optimal flights (e.g., see Castino et al. (2022)) in combination with the seasonal cycle of the tropopause. Results of Yin et al. (2018) (figure 6) show that near the upper cruising altitudes, the seasonal cycle may result in a higher *potcov* for summer than for winter.

6.2 Sensitivity to Threshold Time Until Sunrise

This section will discuss how the results of Chapter 4 answer to the second objective of the thesis: to assess the variability of the contrail ATR20 of eco-efficient trajectories when considering a change in threshold time until sunrise for nighttime contrails. The sensitivity of the results to the selected values for $TH_{sunrise}$ were presented in Section 4.2. This section showed in most cases that the results of this thesis are robust with respect to a varying $TH_{sunrise}$. Most induced variations were small and non-escalating. Conclusions that are drawn from the results in this thesis will not be significantly altered by a change in $TH_{sunrise}$. However, one should be careful when summer day and night mitigation are compared: their relative eco-efficiency does depend on the value of $TH_{sunrise}$.

That the results are robust with a changing $TH_{sunrise}$, also gives an indication of the sensitivity to the assumed lifetime of a persistent contrail. Contrail lifetime affects the contrail climate impact in multiple ways. For example, the spread of a contrail, the distance drifted due to wind and naturally the duration of impact on the radiation balance are depending on contrail lifetime. However, most relevant to this thesis is that changing the assumed contrail lifetime also determines whether a contrail will or will not interact with SW radiation in addition to LW radiation. Since a change in $TH_{sunrise}$ has a similar effect, we conclude that conclusions from this thesis will not be heavily affected if the assumed contrail lifetime would be altered.

The sensitivity results tell us that conditions within 3 and 6 hours to sunrise do not result in significantly different behaviour of the contrail aCCF, neither for the daytime function, nor for the nighttime function. This is not to say that overall results will be unaffected by a changing $TH_{sunrise}$. If $TH_{sunrise}$ is increased, overall (i.e., day + night) results will more and more resemble daytime results. Overall results will be influenced more heavily by nighttime results if a low value for $TH_{sunrise}$ is chosen. To avoid an inappropriate bias to daytime or nighttime in overall results, a $TH_{sunrise}$ should be carefully chosen. Moreover, it is advised to introduce TH_{sunset}, the sunset equivalent of $TH_{sunrise}$. Introducing this threshold (provided that it set to a non-zero value), ensures that a persistent contrail can be treated by ACCF as a nighttime contrail, even though it is formed only shortly before sunset. An additional recommendation for the contrail aCCF is to use half the assumed contrail lifetime for $TH_{sunrise}$ and TH_{sunset}. This has as a consequence that a contrail with over half its lifetime in light is treated as a daytime contrail. Correspondingly, a contrail that has over half its lifetime in darkness is treated as a nighttime contrail.

Alternatively, thrusting in the correctness of the daytime and nighttime contrail aCCF, we advise to use a weight factor while applying them, rather than strictly applying the daytime contrail aCCF or the nighttime contrail aCCF. This makes $TH_{sunrise}$ and TH_{sunset} redundant. The proposed weight factor could be based on the assumed contrail lifetime (t_{life}) and the time until sunrise ($t_{sunrise}$) or sunset (t_{sunset}). If a contrail forms in light, a weight for the daytime and nighttime aCCF (w_{day} $[-]$ and w_{night} $[-]$) can be calculated with equation 6.1 and 6.2, respectively.

$$w_{day} = \frac{t_{sunset}}{t_{life}} \quad \text{with} \quad 0 \leq w_{day} \leq 1 \tag{6.1}$$

$$w_{night} = 1 - w_{day} \tag{6.2}$$

when a contrail forms in darkness, the weight for the nighttime and daytime aCCF can be calculated with equation 6.3 and 6.4.

$$w_{night} = \frac{t_{sunrise}}{t_{life}} \quad \text{with} \quad 0 \leq w_{night} \leq 1 \tag{6.3}$$

$$w_{day} = 1 - w_{night} \tag{6.4}$$

The proposed use of weight factors results in a new version of figure 3.7, which is presented in figure 6.1.

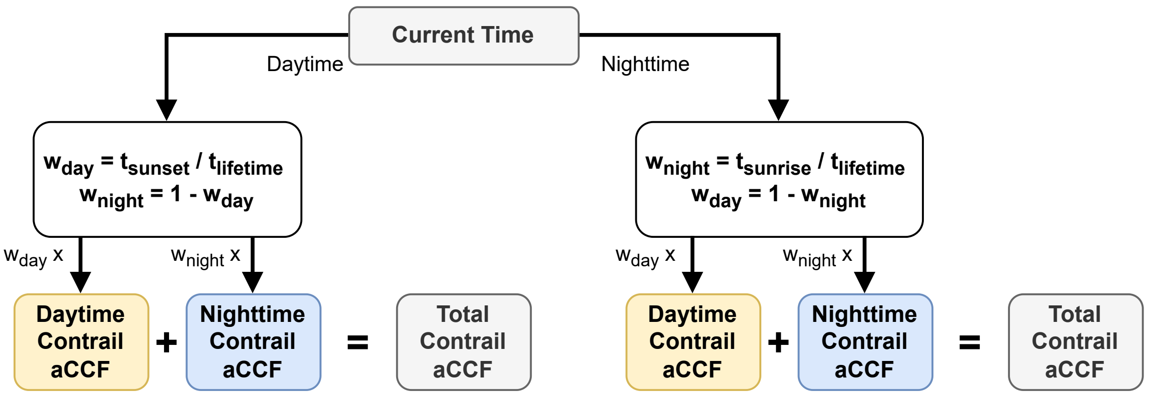

Figure 6.1: Visualisation of proposed new set up for contrail aCCF.

6.3 Benefiting from the Differences Between Day and Night Contrail Climate Impact of Optimised Trajectories

Lastly, this section goes over how the results of Chapter 4 connect to the third objective of the thesis: to explore a climate optimal practice to spend a certain budget of simple operating costs for trajectory optimisation on the minimisation of contrail ATR20. To this end, Section 4.3 presented new solutions that use the difference between daytime and nighttime mitigation to achieve an extra mitigating result. The Section showed that these theoretical solutions indeed exist for both winter and summer. However, the search for new points is more successful in winter than in summer. Most new points make use of a daytime point that is close the the contrail optimal solution, and a nighttime point that is close to the cost optimal solution.

The results for finding new points in the Pareto front contain useful information for policy makers who aim to reduce the climate impact of contrails. Apparently it is rewarding to spend more extra cost on mitigation for daytime flights than on nighttime flights, as this results in a slightly higher reduction of contrail climate impact than when an equal amount of extra costs is spent during both day and night. A policy that aims to reduce the contrail climate impact, should naturally aim for the highest impact reduction at the lowest amount of extra costs. A policy can achieve the cooperation of airliners by making it attractive to optimise trajectories with grants, making it unattractive to not participate by fining uncooperative airliners, or by forcing cooperation (e.g., through closing air space).

We propose a policy that builds upon a minimum required increase in SOC. If an airliner is willing to increase the SOC even further, it should be compensated for its extra efforts. If an airline fails to meet the minimum requirement, the airline should be fined. In the ideal case, the magnitude of compensations and fines should be correlated to the gain or loss in mitigation, rather than the change in SOC. For example, if the extra spent SOC results in a large extra reduction of contrail climate impact, the compensation should be large. If the extra spent SOC results in a minor extra reduction of contrail climate impact, the compensation should be minor as well. Such a policy will encourage airlines to use knee solutions along the Pareto front, and to take advantage of the difference between day and night.

6.4 Assumptions and Uncertainties

The results of this thesis relies on a number of assumptions and is additionally subject to several uncertainties. To properly value the thesis conclusions, these assumptions and uncertainties should be taken into account. This section serves as an overview of the most important assumptions and uncertainties of which the reader should be aware. The section will be subdivided into three categories: uncertainties and assumptions in the modelling chain, uncertainties and assumptions entering through the experiment design and lastly uncertainties and assumptions in the analysis phase. These categories are dealt with in Section 6.4.1, Section 6.4.2 and Section 6.4.3, respectively.

6.4.1 Modelling Chain

The modelling chain applied in this thesis is one of the sources of uncertainty and assumptions. The first source of uncertainty enters through the base model. EMAC is, with the help of ECMWF ERA-Interim reanalysis data, responsible for the simulation of the meteorological fields. Although EMAC is extensively validated (Jöckel et al. (2016)), EMAC remains, as is inherent to a model, a simplification of reality. This simplification results in differences between the modelled atmosphere and the actual atmosphere. For example, some temperature biases are reported by Jöckel et al. (2016). This temperature bias was also observed by Yin et al. (2022). The temperature calculated by EMAC showed values that were on average 3 K lower than the ECMWF reanalysis data. Yin et al. (2022) also shows that the temperature bias affects the predicted *potcov* and the calculated contrail aCCF. Gierens et al. (2020) reported that the meteorological fields simulated by the EMAC model are well capable to predict contrail formation, but the prediction of persistence is *"almost random"*, mainly caused by problems in the prediction of ice supersaturation. The possible biases in the simulated background parameters affect the calculations of all of the three used submodels.

Further sources of uncertainty can be attributed to the submodels. The potential contrail coverage is determined by the CONTRAIL submodel with the help of a set of assumptions. The parameterization of CONTRAIL involves assumptions about subgrid variability of the relative humidity and includes an assumed value for the tuning parameter '*a*' (see equation 3.6) to tune the parameterization. Frömming et al. (2011) validated the CONTRAIL approach but, as mentioned in Section 3.1.3, found an over representation of thin contrails compared to measured data, which may be attributed to either the submodel or the measurements themselves. Nonetheless, Yin et al. (2018) showed that the results of CONTRAIL match well with earlier presented seasonal patterns and spacial distributions of PCC (e.g., Fichter et al. (2005)).

Of all climate affecting agents of aviation, the RF of contrails is characterised by the largest range of uncertainty (Lee et al. (2021)). Inevitably, this uncertainty worked its way into the ACCF submodel and the methods used to derive the aCCF. Moreover, the construction of the contrail aCCF is based on a set of assumptions. For example, following the approach from van Manen and Grewe (2019), an uncertainty arises from the CCF trajectories that are used as meteorological data inputs. The applied Lagrangian trajectories are limited in spacial and temporal representation. This causes uncertainties for the aCCF when they are applied at other geographic locations, other years and other seasons. It should also be mentioned that the original CCF trajectories were simulated by the EMAC model which involves the same uncertainties as earlier described.

Using the method of Schumann et al. (2012), the supplement of Yin et al. (2022) made a set of assumptions towards the construction of contrail aCCFs. A fixed size for ice particles and (as suggested by Schumann et al. (2012)) a size dependent particle mixture is assumed, whilst in reality this size and mixture is far from uniform (Schumann et al. (2017)). Moreover, a fixed contrail depth and width is assumed, which in reality are parameters that differ significantly with contrail age, aircraft type and wind conditions. Additionally, it is mentioned that effects of possible overlapping cloud coverage is disregarded, whilst this actually has an effect on contrail RF.

Additionally, the contrail aCCF itself neglects a large part of the complex effects that drive the radiation imbalance caused by a contrail. While the RF of a contrail depends on a large set of variables, as described in Chapter 2, the formulated contrail aCCF is only dependent on three (or four) of those: T, OLR and $potcov$ (and in a binary way, the time of day). Furthermore, as described in Section 3.1.2, the contrail aCCFs expresses the contrail climate impact in terms of ATR20. This choice introduces an uncertainty as well, as the optical trajectory may vary with the selected climate metric or time horizon, hence uncertainty arises whether a found optimal solution for one climate metric can be harmful in terms of another metric. Matthes et al. (2020) addressed this uncertainty by inspecting the performance of trajectories optimised for ATR20 in terms of eight other metrics (e.g., GWP for a 100 year time horizon). Matthes et al. (2020) found that the ATR20 solutions also reduce climate impact in terms of the other metrics and are thus robust.

Furthermore, one of the major assumptions of Yin et al. (2022) is the contrail lifetime, which is fixed at 6 hours. This assumption does not only affect the contrail lifetime itself, but also affects whether the contrail was treated as daytime or nighttime contrail. The 6 hour lifetime assumption may have a significant effect on the contrail aCCF regarding findings of e.g., Vazquez-Navarro et al. (2015), who found a mean contrail lifetime of approximately 1 hour.

In the application of the contrail aCCF, the ACCF submodel too introduces an uncertainty through the choice in $TH_{sunrise}$. This uncertainty was addressed in this thesis and was found to cause some variation in the results. Nonetheless, the variation did not affect results enough to lead to alternative conclusions.

Several simplifications within the AirTraf submodel also add to the uncertainty of its predictions. The AirTraf aircraft fleet is limited to a few models. Although a representative fleet can be selected from this model set, an AirTraf user should take into account that the actual variety of aircraft models and corresponding performance is not fully captured. Furthermore, the model does not account for trajectory conflicts and can therefore not assure that the proposed set of optimised trajectories is possible in terms of air traffic management. AirTraf computes trajectories with continuous descent or climb rates, rather than the conventional step wise altitude changes between the usual flight levels, and disregards the limits of climb and descent rates. In addition, AirTraf is limited to the cruise phase of a flight and does not account for ground operations, nor does it include the take-off or approach phase of the flight.

6.4.2 Experiment Design

A second source of uncertainty and assumptions is the set up of the experiment for this thesis, which was described in Section 3.2. The experiment has multiple limitations of which one is the limited amount of flown routes. The simulations for this thesis comprised a set of four routes, selected to be diverse in location and orientation over the European continent. Within the budget of four routes this was done well, but contrail ATR20 features a high spacial variability and is therefore sensitive to the simulated routes. To name examples, an inspection of figure 3.6 shows that airspace over France or the Czech Republic is not covered by the route selection. Recall that previous analysis within our research group suggested that three regions with distinct trends regarding the mitigation of contrail ATR20 can be identified within the ATM4E set of routes (Section 3.2.2). The four selected routes fail to include one of these regions, namely Southeast Europe. This region is characterised by little contrail formation in summer and therefore also allows only a very limited amount of contrail ATR20 mitigation. Underrepresentation of south east Europe may thus have resulted in an overestimation of contrail climate impact and the potential to mitigate this. Moreover, one should be careful in transferring the findings of this thesis to other regions than Europe. E.g., more tropical regions are known to feature a different seasonal cycle for contrail formation, making it unsure if our findings hold there too. The limited set of routes has partly been addressed through the verification flight plan, which featured 100 European routes.

The collection of flights of this thesis is also limited in number of temporal instances. No different year than 2018 is accounted for and flights are only simulated for winter and summer for just 30 and 40 days per season, respectively. One should note that the winter of 2018 is not the same as winter in any other year, neither is the summer of 2018 equal to any other summer. Similarly, the end of summer may differ very will from the first summer weeks. Therefore, one should be careful to apply the conclusions of this thesis to days outside those that are simulated. However, it is likely that spring and autumn results will be found in between the summer and winter results, since winter and summer have proven to contain the extremes of the seasonal cycle (e.g., Yin et al. (2018)). Moreover, the verification flight plan again lifts part of the temporal limitations, as the flight plan comprised December, January, February and June, July, August.

Furthermore, Section 3.2.1 describes that all flights depart at 12.00 for the data that are used to select representative days for the simulation. For European flights this means that almost all flight segments will be in daytime. The selection of days is thus representative for flying in daytime, but it is unsure if the chosen days are equally representative for flying in nighttime. Daytime results of this thesis can thus be generalised with more certainty than nighttime results. Nonetheless, nighttime verification results showed a well agreement with the main nighttime results.

Less limited is the chosen set of departure times for the simulations. Comprised of four departures per day, the flight plan ensures that aircraft are in the air for a well spread set of hours in a day. This spread strengthens our confidence that differences between day and night are captured well.

Then, the set of simulations only tests the results for a limited set of three $TH_{sunrise}$ values. The main uncertainty rising from this limited number of different simulations is that we do not know the effect of choosing a negative $TH_{sunrise}$. More precise, we do not know if the nighttime aCCF behaves very differently within a few hours after sunrise. Therefore, the reader should only interpret the results of the $TH_{sunrise}$ variation as an analysis of sensitivity before sunrise.

Moreover, please note that the robustness of the thesis results is rooted in the collective of simulated flights. Due to the high variability of contrail climate impact and mitigation potential, individual flights will rarely show results that are identical to this thesis. Hence, the thesis conclusions should only be applied to a fleet of aircraft or to air traffic associated with a region (like Europe in this work).

Lastly, it is important to note that this work exclusively discusses the mitigation of contrail climate impact. One should be aware that a trajectory or policy that achieves minimal contrail climate impact, may very well be sub-optimal if another climate affecting agent is considered.

6.4.3 Analysis

The analysis phase of this work introduced an additional set of assumptions and uncertainties, which will be described in this section.

Firstly, the day and night distinction for waypoint data is not guaranteed to exactly duplicate the way the contrail aCCF was applied by the ACCF submodel. Inspection of the results showed that wrong classifications happened most frequently for the summer simulations. If wrong classifications do impact the results, this will show as a smaller distinction between night and day, since night will be 'contaminated' with day data and vice versa. An applied correction to wrong classifications only had a minor impact on the generated results. This lead to the believe that wrong classifications are not common enough to cause incorrect conclusions.

Secondly, the day-night distinction flight data makes the assumption that the halfway point and time is a suiting determiner for the time of day as which a flight was classified. In turn, we assumed that the average of the first and last SOC optimal waypoint was a good estimate of the halfway point and time. On top of that, we assumed that the resulting classification holds for the SOC optimal solution, the contrail optimal solution and all solutions in between. This approach is not 100% accurate and may, for example, lead to the nighttime classification of a flight that is 60% in daytime. It is unsure how large the resulting error is. The effect of the error is similar to the error we described above for waypoint data. Nighttime results will be contaminated by daytime results and the other way around, causing a smaller distinction between day and night results.

Towards the exclusion of extra cooling, extra cooling segments were replaced by a more SOC optimal solution. Since the replacing segment originates from a different trajectory, the newly created trajectory is physically impossible. The replacing segment only makes sure that a realistic value for contrail ATR20 and SOC is attributed to the segment. The approach for segment replacing induces an uncertainty on the mitigation results that exclude extra cooling. A physically possible trajectory without extra cooling probably shows a different eco-efficiency. Whether this would result in more ore less eco-efficiency is uncertain.

Lastly, the search for new points within the flight data Pareto front should come with two notes. The first note is that the new points are derived assuming a 50% weight for daytime points and a 50% weight for nighttime points. This means that the corresponding figures are only applicable to an airliner that operates half of its flights during daytime and half of them during nighttime. Alternatively, the figure could apply to a flight that has half of its flight distance during daytime and half during nighttime. If a fleet or flight has a day-night distribution that differs from the 50-50 distribution, a different weight can be applied to calculate new solutions. The second note is that the new points are found offline. AirTraf has thus not simulated the average result of a daytime and a nighttime solution. Although it is likely that a solution exist in the middle, this has not been verified by an actual simulation.

Despite the magnitude and number of uncertainties one should not be discouraged to investigate the possibility of eco-efficient trajectories. As stated by Dahlmann et al. (2016), the reference situation and the mitigation option are to a large extent subjected to the same uncertainties. Thanks to the correlation in error, a robust assessment is still possible. In the same line of thought, following Grewe et al. (2014b), we recognise that within the optimisation, relative values are more important than absolute values.

Chapter 7

Conclusion and Recommendations

This thesis aimed to find the daytime and nighttime contrail climate impact of eco-efficient flight trajectories, to assess the prediction's robustness and to recommend a best practice for eco-efficient flying by using the difference between the daytime and nighttime contrail climate impact. This goal was divided over three objectives:

1. *To analyse the daytime and nighttime difference in contrail ATR20 for a set of eco-efficient trajectories.*
2. *To assess the variability of the contrail ATR20 of eco-efficient trajectories when considering a change in threshold time until sunrise for nighttime contrails.*
3. *To explore a climate optimal practice to spend a certain budget of simple operating costs for trajectory optimisation on the minimisation of contrail ATR20.*

To fulfil these objectives, optimisations and simulations were carried out within the EMAC climate chemistry model, aided by the submodels AirTraf, CONTRAIL and ACCF. AirTraf takes care of the 4D air traffic optimisation comprising a trade-off between minimal Simple Operating Cost (SOC) and minimal contrail climate impact. The quantification of the latter is facilitated by the CONTRAIL and ACCF submodel. The optimisation and simulation were carried out for a selection of 1136 European flights in the winter and summer of 2018. The simulated flights were selected to produce representative results for all European flights in winter and summer. Moreover, the sensitivity of the thesis results were tested against an ACCF model parameter: the threshold time until sunrise ($TH_{sunrise}$). When a contrail is formed during nighttime, this threshold determines whether the contrail is formed sufficiently close to to sunrise to be considered as a daytime contrail. Results were produced for three settings of $TH_{sunrise}$.

Serving the first research objective, the simulation showed that winter daytime mitigation of contrail climate impact is more eco-efficient than winter nighttime mitigation. For summer, daytime mitigation of contrail climate impact has an eco-efficiency that is relatively close to that of nighttime mitigation. We conclude that this difference between the winter and the summer result is a consequence of the wider use of extra cooling contrails for winter mitigation. Overall we observed a trend that daytime mitigation achieves a larger maximum reduction of contrail climate impact than nighttime mitigation and that the daytime advantage largely relies on the formation of extra cooling contrails. Moreover, in absolute terms, summer mitigation is more effective than winter mitigation, driven by the fact that contrail climate impact of cost optimal flights is largest in summer.

Addressing the second research objective, we have seen that above conclusions are robust with respect to a varying $TH_{sunrise}$. As $TH_{sunrise}$ is related to the assumed contrail lifetime, the results also indicate a robustness of the thesis results in case a different lifetime would be assumed. This is not to say that results of daytime and nighttime combined are unaffected by a change in $TH_{sunrise}$. A change in $TH_{sunrise}$ will cause a bias of the overall results to daytime results (large $TH_{sunrise}$) or nighttime results (small or negative $TH_{sunrise}$).

Results for the third objective showed that difference between day and night contrail climate impact mitigation allows for an enhancement of mitigation results. This enhancement appears to be most effective in winter, thanks to the larger difference between day and night mitigation for this season. Moreover, the enhancement of the mitigation of contrail climate impact often makes use of a high amount of extra SOC spend on daytime contrail impact mitigation and a low amount of extra SOC for nighttime contrail impact mitigation.

In light of the Paris agreement, it is recommended to further research the influence of the diurnal and seasonal cycle on the mitigation of contrail climate impact. This work should focus on efforts to explore how these cycles can be used to make the mitigation of aviation's climate impact more effective. The results of this thesis suggest policy makers to encourage daytime mitigation more than nighttime mitigation and to encourage summer mitigation more than winter mitigation.

Based on the results and discussion, it is recommended to repeat several offline analysis methods of this thesis with an online simulation. While results excluding extra cooling contrails were generated offline for this work, a simulation that enforces this restriction online would help verifying dependency of mitigation on extra cooling contrails. Additionally, online verification of research objective three would be of value. This could prove the validity of mitigation enhancement based on the diurnal cycle.

Moreover, we recommend to increase the number of solutions for which waypoint data is stored for analysis. The number of waypoint data points for this thesis (3), allowed for a rough estimation of the shape of the Pareto fronts, but a higher resolution allows for clearer conclusions about the eco-efficiency of the range of solutions. Another recommendation concerning waypoint data is the addition of a day-night marker to the outputs of AirTraf. This will allow for a more accurate distinction between day and night data in the analysis, without discrepancies between ACCF and the analysis method.

Since contrail climate impact and its mitigation is region depended, it is wise to carry out the analysis of this thesis for other regions as well, rather than assuming that the thesis conclusions transferable. The same hold for generating results for other years than 2018: it is recommended to expand the analysis of this thesis to a larger set of years to improve the results robustness. Furthermore, this thesis only accounts for the climate impact of contrails. Further research is necessary to evaluate how the total climate impact of a flight correlates to the thesis results.

Furthermore, validation of the thesis results has proven to be difficult due to a lack of real world data and a scarcity of studies with comparable content. Measurement campaigns of contrail climate impact and the operational mitigation of contrail climate impact could provide the necessary data for validation. Additionally, such campaigns can improve the quality of the ACCF and CONTRAIL submodel and the simulations that are carried out with those submodels.

Data generated for this thesis relies on historical weather data. For future work, it will be interesting to see if trajectory optimisation can also achieve robust mitigation results based on forecasted weather. Therefore it is recommended to perform forecast-based simulations to evaluate if a reliable reduction of climate impact can be achieved.

Lastly, it is recommend to revise the way ACCF distinguishes daytime contrails from nighttime contrails. Instead of marking a contrail either as a daytime contrail or a nighttime contrail, we suggest to apply a weight that corresponds to the fraction of the contrail lifetime in daytime and nighttime, respectively. This avoids that part of the contrail lifetime conflicts with the day-night categorisation by ACCF.

Bibliography

H. Yamashita, V. Grewe, P. Jöckel, F. Linke, M. Schaefer, and D. Sasaki, "Towards climate optimized flight trajectories in a climate model: Airtraf," in *Proceedings of the Eleventh USA/Europe Air Traffic Management Research and Development Seminar (ATM2015), Lisbon, Portugal*, 2015, pp. 23–26.

——, "Air traffic simulation in chemistry-climate model emac 2.41: Airtraf 1.0," *Geoscientific Model Development*, vol. 9, no. 9, pp. 3363–3392, 2016.

F. Yin, V. Grewe, C. Frömming, and H. Yamashita, "Impact on flight trajectory characteristics when avoiding the formation of persistent contrails for transatlantic flights," *Transportation research part D: Transport and environment*, vol. 65, pp. 466–484, 2018.

H. Yamashita, F. Yin, V. Grewe, P. Jöckel, S. Matthes, B. Kern, K. Dahlmann, and C. Frömming, "Newly developed aircraft routing options for air traffic simulation in the chemistry–climate model emac 2.53: Airtraf 2.0," *Geoscientific Model Development*, vol. 13, no. 10, pp. 4869–4890, 2020.

U. Schumann, B. Mayer, K. Graf, and H. Mannstein, "A parametric radiative forcing model for contrail cirrus," *Journal of Applied Meteorology and Climatology*, vol. 51, no. 7, pp. 1391–1406, 2012.

S. Unterstrasser and K. Gierens, "Numerical simulations of contrail-to-cirrus transition–part 1: An extensive parametric study," *Atmospheric Chemistry and Physics*, vol. 10, no. 4, pp. 2017–2036, 2010.

IPCC, *Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*, V. Masson-Delmotte, P. Zhai, A. Pirani, S. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M. Gomis, M. Huang, K. Leitzell, E. Lonnoy, J. Matthews, T. Maycock, T. Waterfield, O. Yelekçi, R. Yu, and B. Zhou, Eds. Cambridge, United Kingdom and New York, NY, USA: Cambridge University Press, 2021.

——, *Climate Change 2022: Impacts, Adaptation, and Vulnerability. Contribution of Working Group II to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*, H.-O. Pörtner, D. Roberts, M. Tignor, E. Poloczanska, K. Mintenbeck, A. Alegría, M. Craig, S. Langsdorf, S. Löschke, V. Möller, A. Okem, and B. Rama, Eds. Cambridge University Press, 2022.

D. S. Lee, D. Fahey, A. Skowron, M. Allen, U. Burkhardt, Q. Chen, S. Doherty, S. Freeman, P. Forster, J. Fuglestvedt *et al.*, "The contribution of global aviation to anthropogenic climate forcing for 2000 to 2018," *Atmospheric Environment*, vol. 244, p. 117834, 2021.

F. Noppel and R. Singh, "Overview on contrail and cirrus cloud avoidance technology," *Journal of Aircraft*, vol. 44, no. 5, pp. 1721–1726, 2007.

M. Narciso and J. M. M. de Sousa, "Influence of sustainable aviation fuels on the formation of contrails and their properties," *Energies*, vol. 14, no. 17, p. 5557, 2021.

T. Bräuer, C. Voigt, D. Sauer, S. Kaufmann, V. Hahn, M. Scheibe, H. Schlager, F. Huber, P. Le Clercq, R. H. Moore *et al.*, "Reduced ice number concentrations in contrails from low-aromatic biofuel blends," *Atmospheric Chemistry and Physics*, vol. 21, no. 22, pp. 16 817–16 826, 2021.

R. Meerkötter, U. Schumann, D. Doelling, P. Minnis, T. Nakajima, and Y. Tsushima, "Radiative forcing by contrails," in *Annales Geophysicae*, vol. 17, no. 8. Springer, 1999, pp. 1080–1094.

C. Fichter, S. Marquart, R. Sausen, and D. S. Lee, "The impact of cruise altitude on contrails and related radiative forcing," *Meteorologische Zeitschrift*, vol. 14, pp. 563–572, 2005.

B. Kärcher, "Formation and radiative forcing of contrail cirrus," *Nature communications*, vol. 9, no. 1, pp. 1–17, 2018.

V. Grewe, T. Champougny, S. Matthes, C. Frömming, S. Brinkop, O. A. Søvde, E. A. Irvine, and L. Halscheidt, "Reduction of the air traffic's contribution to climate change: A react4c case study," *Atmospheric Environment*, vol. 94, pp. 616–625, 2014.

V. Grewe, S. Matthes, C. Frömming, S. Brinkop, P. Jöckel, K. Gierens, T. Champougny, J. Fuglestvedt, A. Haslerud, E. Irvine *et al.*, "Feasibility of climate-optimized air traffic routing for trans-atlantic flights," *Environmental Research Letters*, vol. 12, no. 3, p. 034003, 2017.

S. Matthes, L. Lim, U. Burkhardt, K. Dahlmann, S. Dietmüller, V. Grewe, A. S. Haslerud, J. Hendricks, B. Owen, G. Pitari *et al.*, "Mitigation of non-co2 aviation's climate impact by changing cruise altitudes," *Aerospace*, vol. 8, no. 2, p. 36, 2021.

C. Newinger and U. Burkhardt, "Sensitivity of contrail cirrus radiative forcing to air traffic scheduling," *Journal of Geophysical Research: Atmospheres*, vol. 117, no. D10, 2012.

ATM4E, "Air traffic management for the benefit of environment and climate," https://www.atm4e.eu/index.html, 2018, accessed: 15 February 2022.

FlyATM4E, "Flying air traffic management for the benefit of environment and climate," https://ntrs.nasa.gov/search.jsp?R=19950017884, 2020, accessed: 27 October 2021.

F. Yin, V. Grewe, F. Castino, P. Rao, S. Matthes, K. Dahlmann, S. Dietmüller, C. Frömming, H. Yamashita, P. Peter *et al.*, "Predicting the climate impact of aviation for en-route emissions: The algorithmic climate change function submodel accf 1.0 of emac 2.53," *Geosci. Model Dev. Discuss. [preprint]*, 2022, https://doi.org/10.5194/gmd-2022-220, in review.

U. Burkhardt and B. Kärcher, "Global radiative forcing from contrail cirrus," *Nature climate change*, vol. 1, no. 1, pp. 54–58, 2011.

U. Schumann, R. Baumann, D. Baumgardner, S. T. Bedka, D. P. Duda, V. Freudenthaler, J.-F. Gayet, A. J. Heymsfield, P. Minnis, M. Quante *et al.*, "Properties of individual contrails: a compilation of observations and some comparisons," *Atmospheric Chemistry and Physics*, vol. 17, no. 1, pp. 403–438, 2017.

M. Vazquez-Navarro, H. Mannstein, and S. Kox, "Contrail life cycle and properties from 1 year of msg/seviri rapid-scan images," *Atmospheric Chemistry and Physics*, vol. 15, no. 15, pp. 8739–8749, 2015.

L. Weickmann, "Wolkenbildung durch ein flugzeug," *Naturwissenschaften*, vol. 7, no. 34, pp. 625–625, 1919.

B. Varney, "The argonne battle cloud," *Monthly Weather Review*, vol. 49, no. 6, pp. 348–349, 1921.

J. H. Heiërman, "Vliegtuigwolken," *Hemel en Dampkring*, vol. 42, pp. 101–109, 1944.

A. Brewer, "Condensation trails," *Weather*, vol. 1, no. 2, pp. 34–40, 1946.

A. M. Descamps, "Les traînées blanches d'avions," *Ciel et Terre*, vol. 61, p. 189, 1945.

H. Appleman, "The formation of exhaust condensation trails by jet aircraft," *Bulletin of the American Meteorological Society*, vol. 34, no. 1, pp. 14–20, 1953.

R. Scorer, "Condensation trails," *Weather*, vol. 10, no. 9, pp. 281–287, 1955.

R. Scorer and L. Davenport, “Contrails and aircraft downwash,” *Journal of Fluid Mechanics*, vol. 43, no. 3, pp. 451–464, 1970.

E. Schmidt, “Die entstehung von eisnebel aus den auspuffgasen von flugmotoren,” *Schriften der Deutschen Akademie der Luftfahrtforschung, Verlag R. Oldenbourg, München, Heft 44*, vol. 5, no. 44, pp. 1–15, 1941.

U. Schumann, “On conditions for contrail formation from aircraft exhausts,” *Meteorologische Zeitschrift*, vol. 5, pp. 4–23, 1996.

D. Sonntag, “Advancements in the field of hygrometry,” *Meteorologische Zeitschrift*, pp. 51–66, 1994.

M. L. Schrader, “Calculations of aircraft contrail formation critical temperatures,” *Journal of Applied Meteorology*, vol. 36, no. 12, pp. 1725–1729, 1997.

L. Bock and U. Burkhardt, “The temporal evolution of a long-lived contrail cirrus cluster: Simulations with a global climate model,” *Journal of Geophysical Research: Atmospheres*, vol. 121, no. 7, pp. 3548–3565, 2016.

A. Bier, U. Burkhardt, and L. Bock, “Synoptic control of contrail cirrus life cycles and their modification due to reduced soot number emissions,” *Journal of Geophysical Research: Atmospheres*, vol. 122, no. 21, pp. 11–584, 2017.

M. Bailey and J. Hallett, “Growth rates and habits of ice crystals between- 20 and- 70 c,” *Journal of the Atmospheric Sciences*, vol. 61, no. 5, pp. 514–544, 2004.

F. H. Ludlam, *Clouds and storms: The behavior and effect of water in the atmosphere*. Pennsylvania State University Press, 1980.

A. J. Heymsfield, L. M. Miloshevich, C. Twohy, G. Sachse, and S. Oltmans, “Upper-tropospheric relative humidity observations and implications for cirrus ice nucleation,” *Geophysical research letters*, vol. 25, no. 9, pp. 1343–1346, 1998.

C. Schiller, M. Krämer, A. Afchine, N. Spelten, and N. Sitnikov, “Ice water content of arctic, midlatitude, and tropical cirrus,” *Journal of Geophysical Research: Atmospheres*, vol. 113, no. D24, 2008.

R. Gao, D. Fahey, P. Popp, T. Marcy, R. Herman, E. Weinstock, J. Smith, D. Sayres, J. Pittman, K. Rosenlof *et al.*, “Measurements of relative humidity in a persistent contrail,” *Atmospheric Environment*, vol. 40, no. 9, pp. 1590–1600, 2006.

M. d. Reus, S. Borrmann, A. Bansemer, A. Heymsfield, R. Weigel, C. Schiller, V. Mitev, W. Frey, D. Kunkel, A. Kürten *et al.*, “Evidence for ice particles in the tropical stratosphere from in-situ measurements,” *Atmospheric Chemistry and Physics*, vol. 9, no. 18, pp. 6775–6792, 2009.

J. M. Flores, D. Baumgardner, G. Kok, G. Raga, and R. Hermann, “Tropical subvisual cirrus and contrails at- 85$\circ$ c,” in *12th Conference on Cloud Physics*, 2006, pp. 10–14.

N. Stuber and P. Forster, “The impact of diurnal variations of air traffic on contrail radiative forcing,” *Atmospheric Chemistry and Physics*, vol. 7, no. 12, pp. 3153–3162, 2007.

U. Schumann and K. Graf, “Aviation-induced cirrus and radiation changes at diurnal timescales,” *Journal of Geophysical Research: Atmospheres*, vol. 118, no. 5, pp. 2404–2421, 2013.

R. Sausen, K. Gierens, M. Ponater, and U. Schumann, “A diagnostic study of the global distribution of contrails part i: Present day climate ast,” *Theoretical and Applied Climatology*, vol. 61, no. 3, pp. 127–141, 1998.

D. Avila, L. Sherry, and T. Thompson, “Reducing global warming by airline contrail avoidance: A case study of annual benefits for the contiguous united states,” *Transportation Research Interdisciplinary Perspectives*, vol. 2, p. 100033, 2019.

S. Campbell, N. Neogi, and M. Bragg, "An optimal strategy for persistent contrail avoidance," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008, p. 6515.

B. Zou, G. S. Buxi, and M. Hansen, "Optimal 4-d aircraft trajectories in a contrail-sensitive environment," *Networks and Spatial Economics*, vol. 16, no. 1, pp. 415–446, 2013.

Y. Lim, A. Gardi, and R. Sabatini, "Optimal aircraft trajectories to minimize the radiative impact of contrails and co2," *Energy Procedia*, vol. 110, pp. 446–452, 2017.

H. Yamashita, F. Yin, V. Grewe, P. Jöckel, S. Matthes, B. Kern, K. Dahlmann, and C. Frömming, "Analysis of aircraft routing strategies for north atlantic flights by using airtraf 2.0," *Aerospace*, vol. 8, no. 2, p. 33, 2021.

B. Lührs, F. Linke, S. Matthes, V. Grewe, F. Yin, K. Shine, B. Lührs, and S. Matthes, "Climate impact mitigation potential of european air traffic," in *3rd ECATS conference: Making aviation environmentally sustainable*, 2020.

S. Matthes, B. Lührs, K. Dahlmann, V. Grewe, F. Linke, F. Yin, E. Klingaman, and K. P. Shine, "Climate-optimized trajectories and robust mitigation potential: Flying atm4e," *Aerospace*, vol. 7, no. 11, p. 156, 2020.

B. Lührs, F. Linke, S. Matthes, V. Grewe, and F. Yin, "Climate impact mitigation potential of european air traffic in a weather situation with strong contrail formation," *Aerospace*, vol. 8, no. 2, p. 50, 2021.

H. Yamashita, F. Castino, F. Yin, S. Matthes, V. Grewe, S. Dietmüller, P. Jöckel, and P. Rao, "Multi-objective flight trajectory optimization in emac: Airtraf 3.0," *in preparation, Geoscientific Model Development*, 2023.

P. Jöckel, A. Kerkweg, A. Pozzer, R. Sander, H. Tost, H. Riede, A. Baumgaertner, S. Gromov, and B. Kern, "Development cycle 2 of the modular earth submodel system (messy2)," *Geoscientific Model Development*, vol. 3, no. 2, pp. 717–752, 2010.

E. Roeckner, G. Bäuml, L. Bonaventura, R. Brokopf, M. Esch, M. Giorgetta, S. Hagemann, I. Kirchner, L. Kornblueh, E. Manzini *et al.*, "The atmospheric general circulation model echam 5. part i: Model description," 2003.

E. Roeckner, R. Brokopf, M. Esch, M. Giorgetta, S. Hagemann, L. Kornblueh, E. Manzini, U. Schlese, and U. Schulzweida, "Sensitivity of simulated climate to horizontal and vertical resolution in the echam5 atmosphere model," *Journal of Climate*, vol. 19, no. 16, pp. 3771–3791, 2006.

F. Castino, F. Yin, V. Grewe, H. Yamashita, S. Matthes, S. Baumann, S. Dietmüller, M. Soler, A. Simorgh, F. Linke *et al.*, "Seasonal variability of aircraft trajectories reducing nox-climate impacts under a multitude of weather patterns," 2022.

S. Matthes, "React4c–climate optimised flight planning," in *Innovation for Sustainable Aviation in a Global Environment*. IOS Press, 2012, pp. 122–128.

V. Grewe, C. Frömming, S. Matthes, S. Brinkop, M. Ponater, S. Dietmüller, P. Jöckel, H. Garny, E. Tsati, K. Dahlmann *et al.*, "Aircraft routing with minimal climate impact: The react4c climate cost function modelling approach (v1. 0)," *Geoscientific Model Development*, vol. 7, no. 1, pp. 175–201, 2014.

C. Frömming, V. Grewe, S. Brinkop, P. Jöckel, A. S. Haslerud, S. Rosanka, J. van Manen, and S. Matthes, "Influence of the actual weather situation on non-co2 aviation climate effects: The react4c climate change functions," *Atmospheric Chemistry and Physics Discussions*, 2020.

J. van Manen and V. Grewe, “Algorithmic climate change functions for the use in eco-efficient flight planning,” *Transportation Research Part D: Transport and Environment*, vol. 67, pp. 388–405, 2019.

S. Matthes, V. Grewe, K. Dahlmann, C. Frömming, E. Irvine, L. Lim, F. Linke, B. Lührs, B. Owen, K. Shine *et al.*, “A concept for multi-criteria environmental assessment of aircraft trajectories,” *Aerospace*, vol. 4, no. 3, p. 42, 2017.

J. van Manen, “Aviation h2o and nox climate cost functions based on local weather,” Available online: https : / / repository . tudelft . nl / islandora / object / uuid : 597ed925-9e3b-4300-a2c2-84c8cc97b5b7 ? collection = education, Master thesis, Delft University of Technology, Delft, The Netherlands, April 2017, accessed: 17 September 2020.

E. Irvine, B. Hoskins, and K. Shine, “A lagrangian analysis of ice-supersaturated air over the north atlantic,” *Journal of Geophysical Research: Atmospheres*, vol. 119, no. 1, pp. 90–100, 2014.

U. Schumann, B. Mayer, K. Gierens, S. Unterstrasser, P. Jessberger, A. Petzold, C. Voigt, and J.-F. Gayet, “Effective radius of ice particles in cirrus and contrails,” *Journal of the atmospheric sciences*, vol. 68, no. 2, pp. 300–321, 2011.

U. Burkhardt, B. Kärcher, M. Ponater, K. Gierens, and A. Gettelman, “Contrail cirrus supporting areas in model and observations,” *Geophysical Research Letters*, vol. 35, no. 16, 2008.

K. Gierens, U. Schumann, M. Helten, H. Smit, and A. Marenco, “A distribution law for relative humidity in the upper troposphere and lower stratosphere derived from three years of mozaic measurements,” in *Annales Geophysicae*, vol. 17, no. 9. Copernicus GmbH, 1999, pp. 1218–1226.

C. Frömming, M. Ponater, U. Burkhardt, A. Stenke, S. Pechtl, and R. Sausen, “Sensitivity of contrail coverage and contrail radiative forcing to selected key parameters,” *Atmospheric Environment*, vol. 45, no. 7, pp. 1483–1490, 2011.

G. Myhre, M. Kvalevåg, G. Raedel, J. Cook, K. P. Shine, H. Clark, F. Karcher, K. Markowicz, A. Kardas, P. Wolkenberg *et al.*, “Intercomparison of radiative forcing calculations of stratospheric water vapour and contrails,” *Meteorologische Zeitschrift*, vol. 18, no. 6, p. 585, 2009.

EUROCONTROL, *User manual for the base of aircraft data (BADA) Revision 3.9*, EUROCONTROL, 2011.

F. Deidewig, A. Döpelheuer, and M. Lecht, “Methods to assess aircraft engine emissions in flight,” in *ICAS PROCEEDINGS*, vol. 20, 1996, pp. 131–141.

D. Sasaki and S. Obayashi, “Efficient search for trade-offs by adaptive range multi-objective genetic algorithms,” *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, pp. 44–64, 2005.

M. Burris, “Cost index estimation,” in *IATA 3rd Airline Cost Conference, Geneva, Switzerland*, 2015, pp. 1–23.

C. Frömming, “Documentation of the emac submodels airtrac 1.0 and contrail 1.0, supplementary material of grewe et al., 2014b, geoscientific model development, 7, 175–201,” 2014.

B. Sridhar, N. Y. Chen, and H. K. Ng, “Energy efficient contrail mitigation strategies for reducing the environmental impact of aviation,” in *Tenth USA/Europe Air Traffic Management Research and Development Seminar, ATM2013*, vol. 212, 2013, pp. 1–10.

J. Rosenow, S. Förster, M. Lindner, and H. Fricke, "Impact of multi-critica optimized trajectories on european air traffic density, efficiency and the environment," in *Proceedings of the Twelfth USA/Europe Air Traffic Management Research and Development Seminar, Seattle, DC, USA*, 2017, pp. 26–30.

S. Dietmüller, P. Jöckel, H. Tost, M. Kunze, C. Gellhorn, S. Brinkop, C. Frömming, M. Ponater, B. Steil, A. Lauer *et al.*, "A new radiation infrastructure for the modular earth submodel system (messy, based on version 2.51)," *Geoscientific Model Development*, vol. 9, no. 6, pp. 2209–2222, 2016.

P. Jöckel, H. Tost, A. Pozzer, M. Kunze, O. Kirner, C. A. Brenninkmeijer, S. Brinkop, D. S. Cai, C. Dyroff, J. Eckstein *et al.*, "Earth system chemistry integrated modelling (escimo) with the modular earth submodel system (messy) version 2.51," *Geoscientific Model Development*, vol. 9, no. 3, pp. 1153–1200, 2016.

K. Gierens, S. Matthes, and S. Rohs, "How well can persistent contrails be predicted?" *Aerospace*, vol. 7, no. 12, p. 169, 2020.

K. Dahlmann, V. Grewe, C. Frömming, and U. Burkhardt, "Can we reliably assess climate mitigation options for air traffic scenarios despite large uncertainties in atmospheric processes?" *Transportation Research Part D: Transport and Environment*, vol. 46, pp. 40–55, 2016.

Appendix A

Figures

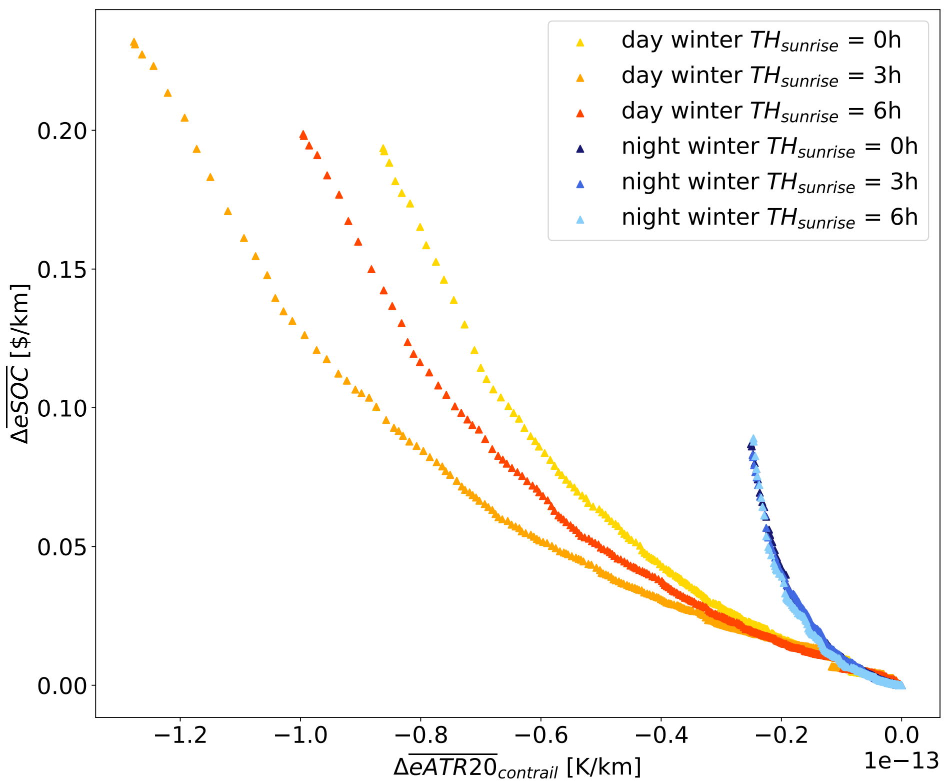

Figure A.1: Pareto front of $\overline{eSOC}$ vs. $\overline{eATR20}_{contrail}$ for day and night in winter generated with flight data, data: $TH_{sunrise}$ = 0h, processed as: $TH_{sunrise}$ = 0, 3 and 6h.

Appendix B

Python Scripts

```python
"""====================================================================================="
" Master file that executes desired thesis functions "
"====================================================================================="""

# Import  functions
from bar_plotter_main import *
from flight_pareto_plotter_main import *
from waypoint_pareto_plotter_main import *
from waypoint_pareto_plotter_no_extra_cooling_main import *
from bar_plotter_vary_t_th import *
from flight_pareto_plotter_vary_t_th import *
from waypoint_pareto_plotter_vary_t_th import *
from waypoint_pareto_plotter_no_extra_cooling_vary_t_th import *

print('##########################################################################################')
print('#################################### STARTING BAR PLOT ####################################')
print('##########################################################################################')
bar_plotter_function(t_th=0)

print('##########################################################################################')
print('############################## STARTING WAYPOINT PARETO PLOT ##############################')
print('##########################################################################################')
waypoint_pareto_plotter_function(t_th=0)

print('##########################################################################################')
print('############################### STARTING FLIGHT PARETO PLOT ###############################')
print('##########################################################################################')
flight_pareto_plotter_function(t_th=0, also_find_new_points=True, also_identify_n_best_new_points=False, n_points=1)

print('##########################################################################################')
print('############################ STARTING WAYPOINT PARETO PLOT NC ############################')
print('##########################################################################################')
```

```python
waypoint_pareto_plotter_no_extra_cooling_function(t_th=0)

print('###########################################################################################')
print('################################ STARTING T_{TH} VARIATION ###############################')
print('###########################################################################################')

print('###########################################################################################')
print('############################### STARTING BAR PLOT VARIATION ###############################')
print('###########################################################################################')
bar_plotter_vary_t_th_function(also_apply_wrong_pp=False)

print('###########################################################################################')
print('######################### STARTING WAYPOINT PARETO PLOT VARIATION #########################')
print('###########################################################################################')
waypoint_pareto_plotter_vary_t_th_function(also_apply_wrong_pp=False)
#
print('###########################################################################################')
print('########################## STARTING FLIGHT PARETO PLOT VARIATION ##########################')
print('###########################################################################################')
flight_pareto_plotter_vary_t_th_function(also_apply_wrong_pp=False)

print('###########################################################################################')
print('####################### STARTING WAYPOINT PARETO PLOT NC VARIATION ########################')
print('###########################################################################################')
waypoint_pareto_plotter_no_extra_cooling_vary_t_th_function(also_apply_wrong_pp=False)

print('###########################################################################################')
print('###################################### DONE WITH ALL ######################################')
print('###########################################################################################')
```

```python
"""=====================================================================================" 
" File that plots the specific contrail ATR20 [K/km] for the SOC optimal solution. The "
" day and night bars are plotted separately for winter and summer. Use is made of "
" waypoint data "
```

```python
"====================================================================================="""

# Import  functions
from bar_plotter_pp_functions import *
import matplotlib.pyplot as plt


def bar_plotter_function(t_th):
    print("BAR PLOTTER FOR t_{TH} = " + str(t_th) + "H STARTED")

    # Define paths to SOC optimal data
    f00_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/Final_PP/W_' + str(t_th) + 'h/f00'
    f00_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/Final_PP/S_' + str(t_th) + 'h/f00'

    # Get specific contrail ATR20 for day and night, winter and summer, SOC optimal solution
    contrail_atr20_results_w = find_specific_day_night_contrail_atr20(file_path=f00_airtraf_folder_w,
                                                                      t_th=t_th)

    specific_day_contrail_atr20_w = contrail_atr20_results_w[0]
    specific_night_contrail_atr20_w = contrail_atr20_results_w[1]
    std_specific_day_contrail_atr20_w = contrail_atr20_results_w[2]
    std_specific_night_contrail_atr20_w = contrail_atr20_results_w[3]

    contrail_atr20_results_s = find_specific_day_night_contrail_atr20(file_path=f00_airtraf_folder_s,
                                                                      t_th=t_th)

    specific_day_contrail_atr20_s = contrail_atr20_results_s[0]
    specific_night_contrail_atr20_s = contrail_atr20_results_s[1]
    std_specific_day_contrail_atr20_s = contrail_atr20_results_s[2]
    std_specific_night_contrail_atr20_s = contrail_atr20_results_s[3]

    print("Winter day   contrail_ATR20 = ", specific_day_contrail_atr20_w.values, " [K/km]")
    print("Winter night contrail_ATR20 = ", specific_night_contrail_atr20_w.values, " [K/km]")
    print("Summer day   contrail_ATR20 = ", specific_day_contrail_atr20_s.values, " [K/km]")
    print("Summer night contrail_ATR20 = ", specific_night_contrail_atr20_s.values, " [K/km]")

    # Plotting
    plt.rcParams.update({'font.size': 20})

    # Collect bars
    day_contrail_atr20_bars = [specific_day_contrail_atr20_w, specific_day_contrail_atr20_s]
    night_contrail_atr20_bars = [specific_night_contrail_atr20_w, specific_night_contrail_atr20_s]

    # Collect whiskers
    day_contrail_atr20_whiskers = [std_specific_day_contrail_atr20_w, std_specific_day_contrail_atr20_s]
    night_contrail_atr20_whiskers = [std_specific_night_contrail_atr20_w, std_specific_night_contrail_atr20_s]

    season = ['Winter', 'Summer']

    fig1, (ax1) = plt.subplots(1, 1, figsize=(10, 7))

    x_axis = np.arange(len(season))
    ax1.yaxis.grid()

    ax1.bar(x_axis - 0.2, day_contrail_atr20_bars, width=0.4, yerr=
    day_contrail_atr20_whiskers, color='gold', edgecolor='white', label='daytime')
    ax1.bar(x_axis + 0.2, night_contrail_atr20_bars, width=0.4, yerr=
    night_contrail_atr20_whiskers, color='midnightblue', edgecolor='white', label='
    nighttime')

    ax1.set_xticks(x_axis)
    ax1.set_xticklabels(season)

    ax1.set_ylabel(r'$\overline{ATR20}_{contrail}$ [K/km]')
    ax1.set_xlabel('Season')

    ax1.legend()

    plt.tight_layout()
    plt.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/Images
    /plots/bar_' + str(t_th) + 'h.pdf')

    plt.show()

    # Get potcov for day and night, winter and summer, SOC optimal solution
    potcov_results_w = find_specific_day_night_potcov(file_path=f00_airtraf_folder_w,
                                                      t_th=t_th)

    specific_day_potcov_w = potcov_results_w[0]
    specific_night_potcov_w = potcov_results_w[1]
    std_specific_day_potcov_w = potcov_results_w[2]
    std_specific_night_potcov_w = potcov_results_w[3]

    potcov_results_s = find_specific_day_night_potcov(file_path=f00_airtraf_folder_s,
                                                      t_th=t_th)

    specific_day_potcov_s = potcov_results_s[0]
    specific_night_potcov_s = potcov_results_s[1]
    std_specific_day_potcov_s = potcov_results_s[2]
    std_specific_night_potcov_s = potcov_results_s[3]

    print("Winter day   potcov = ", specific_day_potcov_w.values, " [km/km]")
    print("Winter night potcov = ", specific_night_potcov_w.values, " [km/km]")
    print("Summer day   potcov = ", specific_day_potcov_s.values, " [km/km]")
    print("Summer night potcov = ", specific_night_potcov_s.values, " [km/km]")

    # Plotting
    plt.rcParams.update({'font.size': 20})

    # Collect bars
    day_potcov_bars = [specific_day_potcov_w, specific_day_potcov_s]
    night_potcov_bars = [specific_night_potcov_w, specific_night_potcov_s]

    # Collect whiskers
    day_potcov_whiskers = [std_specific_day_potcov_w, std_specific_day_potcov_s]
    night_potcov_whiskers = [std_specific_night_potcov_w, std_specific_night_potcov_s]

    season = ['Winter', 'Summer']

    fig, (ax) = plt.subplots(1, 1, figsize=(10, 7))

    x_axis = np.arange(len(season))
    ax.yaxis.grid()

    ax.bar(x_axis - 0.2, day_potcov_bars, width=0.4, yerr=day_potcov_whiskers, color='
    gold', edgecolor='white', label='daytime')
    ax.bar(x_axis + 0.2, night_potcov_bars, width=0.4, yerr=night_potcov_whiskers, color
    ='midnightblue', edgecolor='white', label='nighttime')

    ax.set_xticks(x_axis)
    ax.set_xticklabels(season)

    ax.set_ylabel(r'$potcov$ [km/km]')
    ax.set_xlabel('Season')
    ax.legend()

    plt.tight_layout()
    plt.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/Images
    /plots/potcov_bar_' + str(t_th) + 'h.pdf')

    plt.show()

    return print("BAR PLOTTER FOR t_{TH} = " + str(t_th) + "H FINISHED")
```

```
import numpy as np
import xarray as xr
import suntimes as sts
import datetime as dt
import pandas as pd


def get_per_segment_km(ds):
    """
    ==================================================================================="

    " Function that returns the per segment km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "===================================================================================
    """
    # Get to the segment kilometer property
    ds_km = ds.isel(AirTraf_properties=5).drop_isel(AirTraf_waypoints_out=0)

    return ds_km


def get_per_segment_contrail_atr20(ds):
    """
    ==================================================================================="

    " Function that returns the per segment contrail ATR20 of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "===================================================================================
    """
    # Get to the contrail atr20 property and correct from pulse ATR20 to F-ATR20
    ds_contrail_atr20 = ds.isel(AirTraf_properties=13).drop_isel(AirTraf_waypoints_out
    =0)*0.9654676258992805

    return ds_contrail_atr20


def get_per_segment_potcov(ds):
    """
    ==================================================================================="

    " Function that returns the per segment potcov of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "===================================================================================
    """
    # Get to the potcov property
    ds_potcov = ds.isel(AirTraf_properties=9).drop_isel(AirTraf_waypoints_out=0)

    return ds_potcov


```

```
def get_overall_km(ds):
    """
    ===================================================================================="

    " Function that returns the overall km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "====================================================================================
    """
    # Get to the contrail atr20 property
    ds_km = ds.isel(AirTraf_properties=5)

    # Get overall contrail atr20 by summing over all dimensions
    overall_km = ds_km.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    return overall_km


def get_overall_contrail_atr20(ds):
    """
    ===================================================================================="

    " Function that returns the overall contrail ATR20 of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "====================================================================================
    """
    # Get to the contrail atr20 property and correct from pulse ATR20 to F-ATR20
    ds_contrail_atr20 = ds.isel(AirTraf_properties=13)*0.9654676258992805

    # Get overall contrail atr20 by summing over all dimensions
    overall_contrail_atr20 = ds_contrail_atr20.sum(dim=['AirTraf_waypoints_out', '
    AirTraf_routes_out', 'time'])

    return overall_contrail_atr20


def get_overall_potcov(ds):
    """
    ===================================================================================="

    " Function that returns the overall potcov of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "====================================================================================
    """
    # Get to the potcov property
    ds_potcov = ds.isel(AirTraf_properties=9)

    # Get overall potcov by summing over all dimensions
    overall_potcov = ds_potcov.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', '
    time'])

    return overall_potcov


def get_nighttime_segments(ds_airtraf, t_th):
    """
    ===================================================================================="

    " Function that returns whether a segment is a nighttime (1) or a daytime segment
    (0)    "
    " Inputs: ds_airtraf = AirTraf dataset, t_th = time in hours
```

```
        "

    "====================================================================================
    """

    # ============================== PART I: preparations ==============================
    # Returns the waypoint time stamps in datetime format for riseutc and setutc
    def prepare_timestamps_for_waypoints(time_ns, ref_julian_date_ns):
        # Add the ref_julian_date_ns to ds_time_ns
        julian_date = (time_ns + ref_julian_date_ns)/8.64e13
        # Turn into the datetime format
        time_stamp = pd.to_datetime(julian_date, origin='julian', unit="D")

        return time_stamp

    # Applies prepare_timestamps_for_waypoints to dataset
    def apply_prepare_timestamps_for_waypoints(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        # Get reference date (which is 1 day before first stored data)
        ref_date = ds['time'].values[0] - np.timedelta64(1, 'D')
        # [ns] Convert ref_date to Julian date for easier calculations
        ref_julian_date_ns = pd.Timestamp(ref_date).to_julian_date() * 8.64e13
        # [ns] Convert time data set to Julian date for easier calculations
        ds_time_ns = ds.isel(AirTraf_properties=3) * 8.64e13

        return xr.apply_ufunc(np.vectorize(prepare_timestamps_for_waypoints),
                              ds_time_ns,
                              ref_julian_date_ns,
                              dask='allowed')

    # Applies the sts.Suntimes function to prepare the waypoint place for riseutc and
    setutc
    def prepare_loc_for_suntimes(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        da_lon = ds.isel(AirTraf_properties=0)     # Get the longitude
        da_lat = ds.isel(AirTraf_properties=1)     # Get the latitude
        da_alt = ds.isel(AirTraf_properties=2)     # Get the altitude

        return xr.apply_ufunc(np.vectorize(sts.SunTimes), da_lon, da_lat, da_alt, dask='
    allowed')

    # Returns the segment time stamps in datetime format for get_is_nighttime and
    get_is_nighttime_segment
    def prepare_timestamps_for_segments(time_ns_0, time_ns_1, ref_julian_date_ns):
        # Find the average of the waypoints time
        average_time_ns = (time_ns_1 + time_ns_0) / 2
        # Add the ref_julian_date_ns to ds_time_ns
        average_julian_date = (average_time_ns + ref_julian_date_ns) / 8.64e13
        # Turn into the datetime format
        time_stamp = pd.to_datetime(average_julian_date, origin='julian', unit="D")

        return time_stamp

    # Applies prepare_timestamps_for_segments to dataset
    def apply_prepare_timestamps_for_segments(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        # Get reference date (which is 1 day before first stored data)
        ref_date = ds['time'].values[0] - np.timedelta64(1, 'D')
        # [ns] Convert ref_date to Julian date for easier calculations
        ref_julian_date_ns = pd.Timestamp(ref_date).to_julian_date()*8.64e13
        # [ns] Convert time data set to Julian date for easier calculations
        ds_time_ns = ds.isel(AirTraf_properties=3)*8.64e13
        # Get dataset with last waypoint removed
        ds_time_ns_0 = ds_time_ns.drop_isel(AirTraf_waypoints_out=-1)
        # Get dataset with first waypoint removed
        ds_time_ns_1 = ds_time_ns.drop_isel(AirTraf_waypoints_out=0)

        return xr.apply_ufunc(np.vectorize(prepare_timestamps_for_segments),
                              ds_time_ns_0,
                              ds_time_ns_1,
                              ref_julian_date_ns,
                              dask='allowed')

    # ===================== PART II: get time stamps of current and next-day sunrise
    and sunset ======================
    # Returns the current day time of sunrise with the riseutc function
    def get_time_of_sunrise(loc_prepared, timestamp_prepared, hemisphere='N'):
        date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared)/1e9)
        while True:
            try:
                time_of_sunrise = loc_prepared.riseutc(date_time_timestamp)
            # If in region of polar day or polar night
            except ValueError:
                # If in region and time of polar day
                if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                    # Search backwards for the day when the sun rose
                    date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
                # If in region and time of polar night
                else:
                    # Search forward for the day when the sun will rise
                    date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
                continue
            else:
                break
        return time_of_sunrise

    # Applies get_time_of_sunrise to dataset
    def apply_get_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared):
        return xr.apply_ufunc(np.vectorize(get_time_of_sunrise),
                              ds_loc_prepared,
                              ds_timestamp_prepared,
                              'N',
                              dask='allowed')

    # Returns the current day time of sunset with the setutc function
    def get_time_of_sunset(loc_prepared, timestamp_prepared, hemisphere='N'):
        date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared) / 1e9
    )
        while True:
            try:
                time_of_sunset = loc_prepared.setutc(date_time_timestamp)
            # If in region of polar day or polar night
            except ValueError:
                # If in region and time of polar day
                if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                    # Search forward for the day when the sun will set
                    date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
                # If in region and time of polar night
                else:
                    # Search backward for the day when the sun had set
                    date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
                continue
            else:
                break

        return time_of_sunset

    # Applies get_time_of_sunset to dataset
    def apply_get_time_of_sunset(ds_loc_prepared, ds_timestamp_prepared):
        return xr.apply_ufunc(np.vectorize(get_time_of_sunset),
                              ds_loc_prepared,
                              ds_timestamp_prepared,
                              'N',
                              dask='allowed')

    # Returns the coming day time of sunrise with the riseutc function
    def get_next_day_time_of_sunrise(loc_prepared, timestamp_prepared, hemisphere='N'):
        date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared) / 1e9
    ) + dt.timedelta(days=1)
        while True:
            try:
                next_day_time_of_sunrise = loc_prepared.riseutc(date_time_timestamp)
            # If in region of polar day or polar night
            except ValueError:
                # If in region and time of polar day
                if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                    # Search backward for the day when the sun rose
                    date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
                # If in region and time of polar night
                else:
                    # Search forward for the day when the sun will rise
                    date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
                continue
            else:
                break
        return next_day_time_of_sunrise

    # Applies get_next_day_time_of_sunrise to dataset
    def apply_get_next_day_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared):
        return xr.apply_ufunc(np.vectorize(get_next_day_time_of_sunrise),
                              ds_loc_prepared,
                              ds_timestamp_prepared,
                              'N',
                              dask='allowed')

    # ============================ PART III: get time until sunrise and sunset
    ============================
    # Returns the time until the next sunrise
    def get_time_to_sunrise(timestamp_prepared, t_sunrise, next_day_t_sunrise):
        # If the timestamp is earlier than today's sunrise
        if np.datetime64(timestamp_prepared, 'ns') < np.datetime64(t_sunrise, 'ns'):
            time_to_sunrise = np.datetime64(t_sunrise, 'ns') - np.datetime64(
    timestamp_prepared, 'ns')
        # If today's sunrise has already passed
        else:
            time_to_sunrise = np.datetime64(next_day_t_sunrise, 'ns') - np.datetime64(
    timestamp_prepared, 'ns')

        return time_to_sunrise

    # Applies get_time_to_sunrise to dataset
    def apply_get_time_to_sunrise(ds_timestamp_prepared, ds_t_sunrise,
    ds_next_day_t_sunrise):
        return xr.apply_ufunc(np.vectorize(get_time_to_sunrise),
                              ds_timestamp_prepared,
                              ds_t_sunrise,
                              ds_next_day_t_sunrise,
                              dask='allowed')

    # ============================ PART IV: Check if it is currently nighttime
    ============================
    # Returns whether a waypoints is in nighttime
    # Returns 1 if the current-day sunset has passed or the current-day sunrise is yet
    to come. Else returns 0
    def get_is_night(timestamp_prepared, t_sunrise, t_sunset):
        # If the timestamp is earlier than today's sunrise OR after today's sunset
        if (np.datetime64(timestamp_prepared, 'ns') < np.datetime64(t_sunrise, 'ns')) or
     (np.datetime64(timestamp_prepared, 'ns') > np.datetime64(t_sunset, 'ns')):
            is_night = 1
        else:
            is_night = 0

        return is_night

    # Applies get_is_night to dataset
    def apply_get_is_night(ds_timestamp_segments_prepared, ds_segment_t_sunrise,
    ds_segment_t_sunset):
```

```python
    return xr.apply_ufunc(np.vectorize(get_is_night),
                          ds_timestamp_segments_prepared,
                          ds_segment_t_sunrise,
                          ds_segment_t_sunset,
                          dask='allowed')

# ============================== PART V: Check if the segment is a nighttime segment ==============================
# Returns whether a waypoint is a nighttime waypoint
# Returns 1 if it is night and the time to sunrise is larger than the threshold time to sunrise. Else returns 0
def get_is_night_segment(nighttime, time_to_sunrise, t_th, contrail_atr20):
    # if the contrail ATR20 has a value smaller than zero it is definitely daytime
    if contrail_atr20 < 0:
        is_night_segment = 0*nighttime
    # If the time until sunrise is larger than the given threshold time to sunrise
    elif np.timedelta64(time_to_sunrise, 'ns') > np.timedelta64(int(t_th * 3.6e12), 'ns'):
        is_night_segment = 1*nighttime
    else:
        is_night_segment = 0*nighttime

    return is_night_segment

# Applies  get_is_night_segment to dataset
def apply_get_is_night_segment(ds_nighttime, ds_segment_time_to_sunrise, t_th, ds_contrail_atr20):
    return xr.apply_ufunc(np.vectorize(get_is_night_segment),
                          ds_nighttime,
                          ds_segment_time_to_sunrise,
                          t_th,
                          ds_contrail_atr20,
                          dask='allowed')

# ============================== PART VI: Execute the embedded functions ==============================
# Part I
ds_timestamp_prepared = apply_prepare_timestamps_for_waypoints(ds_airtraf)
ds_loc_prepared = prepare_loc_for_suntimes(ds_airtraf)
ds_timestamp_segments_prepared = apply_prepare_timestamps_for_segments(ds_airtraf)

# Part II
ds_t_sunrise = apply_get_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared)
ds_t_sunset = apply_get_time_of_sunset(ds_loc_prepared, ds_timestamp_prepared)
ds_next_day_t_sunrise = apply_get_next_day_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared)

# Find the time of sunrise and sunset per segment by averaging the time of its end waypoints
ds_segment_t_sunrise = ds_t_sunrise.drop_isel(AirTraf_waypoints_out=-1)+(ds_t_sunrise.drop_isel(AirTraf_waypoints_out=0)-ds_t_sunrise.drop_isel(AirTraf_waypoints_out=-1))/2
ds_segment_t_sunset = ds_t_sunset.drop_isel(AirTraf_waypoints_out=-1)+(ds_t_sunset.drop_isel(AirTraf_waypoints_out=0)-ds_t_sunset.drop_isel(AirTraf_waypoints_out=-1))/2

# Part III
ds_time_to_sunrise = apply_get_time_to_sunrise(ds_timestamp_prepared, ds_t_sunrise, ds_next_day_t_sunrise)
ds_segment_time_to_sunrise = (ds_time_to_sunrise.drop_isel(AirTraf_waypoints_out=-1)+ds_time_to_sunrise.drop_isel(AirTraf_waypoints_out=0))/2

# Part IV
ds_nighttime = apply_get_is_night(ds_timestamp_segments_prepared, ds_segment_t_sunrise, ds_segment_t_sunset)

# Part V
ds_nighttime_segments = apply_get_is_night_segment(ds_nighttime, ds_segment_time_to_sunrise, t_th, ds_airtraf["routes_out"].isel(AirTraf_properties
    =13).drop_isel(AirTraf_waypoints_out=0))

    return ds_nighttime_segments


def find_specific_day_night_contrail_atr20(file_path, t_th):
    """
    ====================================================================================="
    " Function that returns the specific contrail ATR20 separately for day and night
        "
    " Inputs: file_path = folder with the *airtraf_ac.nc file(s)
        "

    "=====================================================================================
    """

    # Open the airtraf_ac.nc file
    ds_airtraf = xr.open_mfdataset('{}*airtraf_ac.nc'.format(file_path))

    # Correct for 12 h time step by removing duplicates if necessary
    if ds_airtraf['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf = ds_airtraf.drop_isel(time=np.arange(0, len(ds_airtraf["routes_out"
    ]), 2).tolist())

    # Get the overall contrail ATR20
    overall_contrail_atr20 = get_overall_contrail_atr20(ds_airtraf["routes_out"])
    # Get the overall km
    overall_km = get_overall_km(ds_airtraf["routes_out"])

    # Get per segment contrail ATR20
    per_segment_contrail_atr20 = get_per_segment_contrail_atr20(ds_airtraf["routes_out"
    ])
    # Get the per_segment km
    per_segment_km = get_per_segment_km(ds_airtraf["routes_out"])

    # Get witch segments are during nighttime
    nighttime_segments = get_nighttime_segments(ds_airtraf, t_th)

    # Apply night mask over per_segment_contrail_atr20
    per_segment_night_contrail_atr20 = per_segment_contrail_atr20 * nighttime_segments
    # Apply night mask over per_segment_km
    per_segment_night_km = per_segment_km * nighttime_segments

    # Sum to obtain the total night contrail ATR20
    night_contrail_atr20 = per_segment_night_contrail_atr20.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    # Sum to obtain the total night km
    night_km = per_segment_night_km.sum(dim=['AirTraf_waypoints_out', '
    AirTraf_routes_out', 'time'])

    # Find the the total day contrail ATR20 by difference
    day_contrail_atr20 = overall_contrail_atr20 - night_contrail_atr20
    # Find the the total day km by difference
    day_km = overall_km - night_km

    # Find night specific contrail ATR20
    specific_night_contrail_atr20 = night_contrail_atr20 / night_km
    # Find day specific contrail ATR20
    specific_day_contrail_atr20 = day_contrail_atr20 / day_km

    daytime_segments = (nighttime_segments - 1)*-1
    prep_std_nighttime_segments = nighttime_segments.where(nighttime_segments != 0, np.
    nan)
    prep_std_daytime_segments = daytime_segments.where(daytime_segments != 0, np.nan)
    prep_std_per_segment_night_contrail_atr20 = per_segment_contrail_atr20 *
    prep_std_nighttime_segments
    prep_std_per_segment_night_km = per_segment_km * prep_std_nighttime_segments
    prep_std_per_segment_day_contrail_atr20 = per_segment_contrail_atr20 *
    prep_std_daytime_segments
    prep_std_per_segment_day_km = per_segment_km * prep_std_daytime_segments

    prep_std_specific_night_contrail_atr20 = prep_std_per_segment_night_contrail_atr20 /
     prep_std_per_segment_night_km
    prep_std_specific_day_contrail_atr20 = prep_std_per_segment_day_contrail_atr20 /
    prep_std_per_segment_day_km

    std_specific_night_contrail_atr20 = prep_std_specific_night_contrail_atr20.mean(dim
    =["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True).std()
    std_specific_day_contrail_atr20 = prep_std_specific_day_contrail_atr20.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True).std()

    to_be_returned = (specific_day_contrail_atr20,
                      specific_night_contrail_atr20,
                      std_specific_day_contrail_atr20,
                      std_specific_night_contrail_atr20)

    return to_be_returned


def find_specific_day_night_potcov(file_path, t_th):
    """
    ====================================================================================="

    " Function that returns the specific potcov separately for day and night
        "
    " Inputs: file_path = folder with the *airtraf_ac.nc file(s)
        "

    "=====================================================================================
    """

    # Open the airtraf_ac.nc file
    ds_airtraf = xr.open_mfdataset('{}*airtraf_ac.nc'.format(file_path))

    # Correct for 12 h time step by removing duplicates if necessary
    if ds_airtraf['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf = ds_airtraf.drop_isel(time=np.arange(0, len(ds_airtraf["routes_out"
    ]), 2).tolist())

    # Get the overall potcov
    overall_potcov = get_overall_potcov(ds_airtraf["routes_out"])
    # Get the overall km
    overall_km = get_overall_km(ds_airtraf["routes_out"])

    # Get per segment potcov
    per_segment_potcov = get_per_segment_potcov(ds_airtraf["routes_out"])
    # Get the per_segment km
    per_segment_km = get_per_segment_km(ds_airtraf["routes_out"])

    # Get witch segments are during nighttime
    nighttime_segments = get_nighttime_segments(ds_airtraf, t_th)

    # Apply night mask over per_segment_potcov
    per_segment_night_potcov = per_segment_potcov * nighttime_segments
    # Apply night mask over per_segment_km
    per_segment_night_km = per_segment_km * nighttime_segments

    # Sum to obtain the total night potcov
    night_potcov = per_segment_night_potcov.sum(dim=['AirTraf_waypoints_out',
                                                     'AirTraf_routes_out',
                                                     'time'])
    # Sum to obtain the total night km
    night_km = per_segment_night_km.sum(dim=['AirTraf_waypoints_out',
                                             'AirTraf_routes_out',
                                             'time'])

    # Find the the total day contrail ATR20 by difference
    day_potcov = overall_potcov - night_potcov
    # Find the the total day km by difference
    day_km = overall_km - night_km

    # Find night specific potcov
    specific_night_potcov = night_potcov / night_km
    # Find day specific potcov
    specific_day_potcov = day_potcov / day_km

    daytime_segments = (nighttime_segments - 1) * -1
    prep_std_nighttime_segments = nighttime_segments.where(nighttime_segments != 0, np.
    nan)
    prep_std_daytime_segments = daytime_segments.where(daytime_segments != 0, np.nan)
    prep_std_per_segment_night_potcov = per_segment_potcov * prep_std_nighttime_segments
    prep_std_per_segment_night_km = per_segment_km * prep_std_nighttime_segments
    prep_std_per_segment_day_potcov = per_segment_potcov * prep_std_daytime_segments
    prep_std_per_segment_day_km = per_segment_km * prep_std_daytime_segments

    prep_std_specific_night_potcov = prep_std_per_segment_night_potcov /
    prep_std_per_segment_night_km
    prep_std_specific_day_potcov = prep_std_per_segment_day_potcov /
    prep_std_per_segment_day_km

    std_specific_night_potcov = prep_std_specific_night_potcov.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True).std()
    std_specific_day_potcov = prep_std_specific_day_potcov.mean(dim=["AirTraf_routes_out
    ", "AirTraf_waypoints_out"], skipna=True).std()

    to_be_returned = (specific_day_potcov,
                      specific_night_potcov,
                      std_specific_day_potcov,
                      std_specific_night_potcov)

    return to_be_returned
```

```
"""
    ======================================================================================"
" File that plots the delta effective specific contrail ATR20 [K/km] as Pareto front.
    "
" The day and night front are plotted separately for winter and summer. Use is made of
    "
" waypoint data.
    "
"=====================================================================================
    """

# Import functions
from waypoint_pareto_plotter_pp_functions import *
import matplotlib.pyplot as plt


def waypoint_pareto_plotter_function(t_th):
    print("WAYPOINT DATA PARETO PLOTTER FOR t_{TH} = " + str(t_th) + "H STARTED")

    # Define paths to data
    f00_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/W_' + str(t_th) + 'h/f00'
    f05_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/W_' + str(t_th) + 'h/f05'
    f100_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes
    /Final_PP/W_' + str(t_th) + 'h/f100'
    f00_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/S_' + str(t_th) + 'h/f00'
    f05_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/S_' + str(t_th) + 'h/f05'
    f100_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes
    /Final_PP/S_' + str(t_th) + 'h/f100'

    print("[[[[[[[[[[[[[[[[[[[[[[[WINTER]]]]]]]]]]]]]]]]]]]]]]]")
    # Get delta effective specific contrail ATR20 for day and night, winter and summer,
    SOC optimal solution, +0.5% SOC solution and climate optimal solution
    results_w = get_average_pareto_front_waypoints(f00_airtraf_folder=
    f00_airtraf_folder_w,
                                                   f05_airtraf_folder=
    f05_airtraf_folder_w,
                                                   f100_airtraf_folder=
    f100_airtraf_folder_w,
                                                   t_th=t_th)

    delta_daytime_contrail_atr20_lst_w = results_w[0]
    delta_nighttime_contrail_atr20_lst_w = results_w[1]
    delta_daytime_soc_lst_w = results_w[2]
    delta_nighttime_soc_lst_w = results_w[3]
    func_daytime_w = results_w[4]
    func_nighttime_w = results_w[5]
    max_nighttime_relative_change_contrail_atr20_w = results_w[6]
    max_nighttime_relative_change_soc_w = results_w[7]
    max_daytime_relative_change_contrail_atr20_w = results_w[8]
    max_daytime_relative_change_soc_w = results_w[9]
    nighttime_segments_00_w = results_w[10]
    nighttime_segments_05_w = results_w[11]
    nighttime_segments_100_w = results_w[12]
    std_plus_delta_nighttime_contrail_atr20_lst_w = results_w[13]
    std_plus_delta_daytime_contrail_atr20_lst_w = results_w[14]
    std_plus_delta_nighttime_soc_lst_w = results_w[15]
    std_plus_delta_daytime_soc_lst_w = results_w[16]
    std_min_delta_nighttime_contrail_atr20_lst_w = results_w[17]
    std_min_delta_daytime_contrail_atr20_lst_w = results_w[18]
    std_min_delta_nighttime_soc_lst_w = results_w[19]
    std_min_delta_daytime_soc_lst_w = results_w[20]
    std_plus_func_nighttime_w = results_w[21]
    std_plus_func_daytime_w = results_w[22]
    std_min_func_nighttime_w = results_w[23]
    std_min_func_daytime_w = results_w[24]
    med_nighttime_relative_change_contrail_atr20_w = results_w[25]
    med_nighttime_relative_change_soc_w = results_w[26]
    med_daytime_relative_change_contrail_atr20_w = results_w[27]
    med_daytime_relative_change_soc_w = results_w[28]

    get_potcov_average_pareto_front_waypoints(f00_airtraf_folder=f00_airtraf_folder_w,
                                              f05_airtraf_folder=f05_airtraf_folder_w,
                                              f100_airtraf_folder=f100_airtraf_folder_w,
                                              nighttime_segments_00=
    nighttime_segments_00_w,
                                              nighttime_segments_05=
    nighttime_segments_05_w,
                                              nighttime_segments_100=
    nighttime_segments_100_w)

    print("[[[[[[[[[[[[[[[[[[[[[[[SUMMER]]]]]]]]]]]]]]]]]]]]]]]")
    results_s = get_average_pareto_front_waypoints(f00_airtraf_folder=
    f00_airtraf_folder_s,
                                                   f05_airtraf_folder=
    f05_airtraf_folder_s,
                                                   f100_airtraf_folder=
    f100_airtraf_folder_s,
                                                   t_th=t_th)

    delta_daytime_contrail_atr20_lst_s = results_s[0]
    delta_nighttime_contrail_atr20_lst_s = results_s[1]
    delta_daytime_soc_lst_s = results_s[2]
    delta_nighttime_soc_lst_s = results_s[3]
    func_daytime_s = results_s[4]
    func_nighttime_s = results_s[5]
    max_nighttime_relative_change_contrail_atr20_s = results_s[6]
    max_nighttime_relative_change_soc_s = results_s[7]
    max_daytime_relative_change_contrail_atr20_s = results_s[8]
    max_daytime_relative_change_soc_s = results_s[9]
    nighttime_segments_00_s = results_s[10]
    nighttime_segments_05_s = results_s[11]
    nighttime_segments_100_s = results_s[12]
    std_plus_delta_nighttime_contrail_atr20_lst_s = results_s[13]
    std_plus_delta_daytime_contrail_atr20_lst_s = results_s[14]
    std_plus_delta_nighttime_soc_lst_s = results_s[15]
    std_plus_delta_daytime_soc_lst_s = results_s[16]
    std_min_delta_nighttime_contrail_atr20_lst_s = results_s[17]
    std_min_delta_daytime_contrail_atr20_lst_s = results_s[18]
    std_min_delta_nighttime_soc_lst_s = results_s[19]
    std_min_delta_daytime_soc_lst_s = results_s[20]
    std_plus_func_nighttime_s = results_s[21]
    std_plus_func_daytime_s = results_s[22]
    std_min_func_nighttime_s = results_s[23]
    std_min_func_daytime_s = results_s[24]
    med_nighttime_relative_change_contrail_atr20_s = results_s[25]
    med_nighttime_relative_change_soc_s = results_s[26]
    med_daytime_relative_change_contrail_atr20_s = results_s[27]
    med_daytime_relative_change_soc_s = results_s[28]

    get_potcov_average_pareto_front_waypoints(f00_airtraf_folder=f00_airtraf_folder_s,
                                              f05_airtraf_folder=f05_airtraf_folder_s,
                                              f100_airtraf_folder=f100_airtraf_folder_s,
                                              nighttime_segments_00=
    nighttime_segments_00_s,
                                              nighttime_segments_05=
    nighttime_segments_05_s,
                                              nighttime_segments_100=
    nighttime_segments_100_s)

    # Plotting
    plt.rcParams.update({'font.size': 20})
    fig, (ax) = plt.subplots(1, 1, figsize=(12, 10))

    ax.scatter(delta_daytime_contrail_atr20_lst_w, delta_daytime_soc_lst_w, marker='^',
    color='gold', label='day winter')
    ax.scatter(delta_nighttime_contrail_atr20_lst_w, delta_nighttime_soc_lst_w, marker='
    ^', color='midnightblue', label='night winter')
    ax.plot(delta_nighttime_contrail_atr20_lst_w, func_nighttime_w(
    delta_nighttime_contrail_atr20_lst_w), color='midnightblue', linestyle='dashed')
    ax.plot(delta_daytime_contrail_atr20_lst_w, func_daytime_w(
    delta_daytime_contrail_atr20_lst_w), color='gold', linestyle='dashed')
    ax.fill(np.append(std_min_delta_nighttime_contrail_atr20_lst_w,
    std_plus_delta_nighttime_contrail_atr20_lst_w[::-1]),
            np.append(delta_nighttime_soc_lst_w, delta_nighttime_soc_lst_w[::-1]),
            color='midnightblue',
            alpha=0.1)
    ax.fill(np.append(std_min_delta_daytime_contrail_atr20_lst_w,
    std_plus_delta_daytime_contrail_atr20_lst_w[::-1]),
            np.append(delta_daytime_soc_lst_w, delta_daytime_soc_lst_w[::-1]),
            color='gold',
            alpha=0.1)

    ax.scatter(delta_daytime_contrail_atr20_lst_s, delta_daytime_soc_lst_s, marker='o',
    color='gold', label='day summer')
    ax.scatter(delta_nighttime_contrail_atr20_lst_s, delta_nighttime_soc_lst_s, marker='
    o', color='midnightblue', label='night summer')
    ax.plot(delta_nighttime_contrail_atr20_lst_s, func_nighttime_s(
    delta_nighttime_contrail_atr20_lst_s), color='midnightblue', linestyle='dashed')
    ax.plot(delta_daytime_contrail_atr20_lst_s, func_daytime_s(
    delta_daytime_contrail_atr20_lst_s), color='gold', linestyle='dashed')
    ax.fill(np.append(std_min_delta_nighttime_contrail_atr20_lst_s,
    std_plus_delta_nighttime_contrail_atr20_lst_s[::-1]),
            np.append(delta_nighttime_soc_lst_s, delta_nighttime_soc_lst_s[::-1]),
            color='midnightblue',
            alpha=0.1)
    ax.fill(np.append(std_min_delta_daytime_contrail_atr20_lst_s,
    std_plus_delta_daytime_contrail_atr20_lst_s[::-1]),
            np.append(delta_daytime_soc_lst_s, delta_daytime_soc_lst_s[::-1]),
            color='gold',
            alpha=0.1)

    ax.set_xlabel(r'$\Delta \overline{eATR20}_{contrail}$ [K/km]')
    ax.set_ylabel(r'$\Delta \overline{eSOC}$ [\$/km]')
    ax.legend()

    ax.text(min(delta_daytime_contrail_atr20_lst_w), max(delta_daytime_soc_lst_w), '('+
    str(max_daytime_relative_change_contrail_atr20_w)+'%,'+str(
    max_daytime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(min(delta_nighttime_contrail_atr20_lst_w), max(delta_nighttime_soc_lst_w), '
    ('+str(max_nighttime_relative_change_contrail_atr20_w)+'%,'+str(
    max_nighttime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')
    ax.text(min(delta_daytime_contrail_atr20_lst_s), max(delta_daytime_soc_lst_s), '('+
    str(max_daytime_relative_change_contrail_atr20_s)+'%,'+str(
    max_daytime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(min(delta_nighttime_contrail_atr20_lst_s), max(delta_nighttime_soc_lst_s), '
    ('+str(max_nighttime_relative_change_contrail_atr20_s)+'%,'+str(
    max_nighttime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')

    ax.text(delta_daytime_contrail_atr20_lst_w[1], delta_daytime_soc_lst_w[1], '('+str(
    med_daytime_relative_change_contrail_atr20_w)+'%,'+str(
    med_daytime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(delta_nighttime_contrail_atr20_lst_w[1], delta_nighttime_soc_lst_w[1], '('+
    str(med_nighttime_relative_change_contrail_atr20_w)+'%,'+str(
    med_nighttime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')
    ax.text(delta_daytime_contrail_atr20_lst_s[1], delta_daytime_soc_lst_s[1], '('+str(
    med_daytime_relative_change_contrail_atr20_s)+'%,'+str(
    med_daytime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(delta_nighttime_contrail_atr20_lst_s[1], delta_nighttime_soc_lst_s[1], '('+
    str(med_nighttime_relative_change_contrail_atr20_s)+'%,'+str(
    med_nighttime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')

    plt.tight_layout()
    fig.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/Images
    /plots/waypoint_pareto_' + str(t_th) + 'h.pdf')

    plt.show()

    return print("WAYPOINT DATA PARETO PLOTTER FOR t_{TH} = " + str(t_th) + "H FINISHED"
    )
```

```
import numpy as np
import xarray as xr
import suntimes as sts
import datetime as dt
import pandas as pd


def get_per_segment_km(ds):
    """
    ========================================================================================"

    " Function that returns the per segment km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "========================================================================================
    """
    # Get to the segment kilometer property
    ds_km = ds.isel(AirTraf_properties=5).drop_isel(AirTraf_waypoints_out=0)
```

```python

    return ds_km


def get_effective_per_segment_km(ds):
    """
    ===================================================================================="
    " Function that returns the  effective per segment km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "====================================================================================
    """
    # Find the jump in longitude and latitude for each segment
    segment_lon_1 = ds.isel(AirTraf_properties=0).drop_isel(AirTraf_waypoints_out=-1)
    segment_lon_2 = ds.isel(AirTraf_properties=0).drop_isel(AirTraf_waypoints_out=0)
    segment_delta_lon = segment_lon_2 - segment_lon_1
    segment_lat_1 = ds.isel(AirTraf_properties=1).drop_isel(AirTraf_waypoints_out=-1)
    segment_lat_2 = ds.isel(AirTraf_properties=1).drop_isel(AirTraf_waypoints_out=0)
    segment_delta_lat = segment_lat_2 - segment_lat_1

    # prevent later division by zero
    segment_delta_lon_safe = segment_delta_lon.where(abs(segment_delta_lon) > 0.00001,
    0.00001)

    # Find the jump in longitude and latitude for each flight
    flight_lon_1 = ds.isel(AirTraf_properties=0).isel(AirTraf_waypoints_out=0)
    flight_lon_2 = ds.isel(AirTraf_properties=0).isel(AirTraf_waypoints_out=-1)
    flight_delta_lon = flight_lon_2 - flight_lon_1
    flight_lat_1 = ds.isel(AirTraf_properties=1).isel(AirTraf_waypoints_out=0)
    flight_lat_2 = ds.isel(AirTraf_properties=1).isel(AirTraf_waypoints_out=-1)
    flight_delta_lat = flight_lat_2 - flight_lat_1

    # prevent later division by zero
    flight_delta_lon_safe = flight_delta_lon.where(abs(flight_delta_lon) > 0.00001,
    0.00001)

    # Take the arctan of the jump in latitude over the jump in longitude to get the
    angle
    segment_orientation = np.arctan(segment_delta_lat / segment_delta_lon_safe)
    flight_orientation = np.arctan(flight_delta_lat / flight_delta_lon_safe)

    # Find the difference in angle
    delta_orientation = abs(segment_orientation - flight_orientation)

    # Find the corresponding cosine
    segment_cos = np.cos(delta_orientation)

    # Get the plain distance
    per_segment_km = get_per_segment_km(ds)

    # Compute the effective component of the plain distance
    effective_distance = segment_cos * per_segment_km

    return effective_distance


def get_per_segment_contrail_atr20(ds):
    """
    ===================================================================================="
    " Function that returns the per segment contrail ATR20 of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "====================================================================================
    """
    # Get to the contrail atr20 property and correct from pulse ATR20 to F-ATR20
    ds_contrail_atr20 = ds.isel(AirTraf_properties=13).drop_isel(AirTraf_waypoints_out
    =0) * 0.9654676258992805

    return ds_contrail_atr20


def get_per_segment_potcov(ds):
    """
    ====================================================================================="

    " Function that returns the per segment potcov of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "=====================================================================================
    """
    # Get to the potcov property
    ds_potcov = ds.isel(AirTraf_properties=9).drop_isel(AirTraf_waypoints_out=0)

    return ds_potcov


def get_per_segment_soc(ds):
    """
    ====================================================================================="

    " Function that returns the per segment Simple Operating Costs of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "=====================================================================================
    """
    fuel_price = 1.545                      # Average fuel Price in March 2017 [US Dollar/
    US Gallon]
    fuel_density = 6.71                     # Fuel_density [lbs/US Gallon]
    c_f_lbs = fuel_price/fuel_density       # Unit fuel costs [US Dollar/lbs]
    c_f_kg = c_f_lbs/0.45359237             # Unit fuel costs [US Dollar/kg]

    c_t_h = 2710                            # Unit time costs [US Dollar/h]
    c_o = 0.0                               # other costs [US Dollar]

    ds_v = ds.isel(AirTraf_properties=4, AirTraf_waypoints_out=slice(0, -1))    #
    Aircraft ground speed [km/h]
    ds_d = ds.isel(AirTraf_properties=5, AirTraf_waypoints_out=slice(1, None))  #
    Distance for each segment [km]
    ds_f = ds.isel(AirTraf_properties=6, AirTraf_waypoints_out=slice(1, None))  # Fuel
    use [kg]

    # Calculate time by dividing distance over speed
    ds_t = ds_d/ds_v

    # Calculate soc per segment in $
    ds_soc = c_t_h * ds_t + c_f_kg * ds_f + c_o

    return ds_soc


def get_overall_km(ds):
    """
    ====================================================================================="

    " Function that returns the overall km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "====================================================================================="""
    # Get to the distance property
    ds_km = ds.isel(AirTraf_properties=5)

    # Get overall contrail atr20 by summing over all dimensions
    overall_km = ds_km.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    return overall_km


def get_overall_contrail_atr20(ds):
    """====================================================================================="
    " Function that returns the overall contrail ATR20 of a data set    "
    " Inputs: ds = AirTraf dataset routes_out    "
    "====================================================================================="""
    # Get to the contrail atr20 property and correct from pulse ATR20 to F-ATR20
    ds_contrail_atr20 = ds.isel(AirTraf_properties=13) * 0.9654676258992805

    # Get overall contrail atr20 by summing over all dimensions
    overall_contrail_atr20 = ds_contrail_atr20.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    return overall_contrail_atr20


def get_overall_potcov(ds):
    """====================================================================================="
    " Function that returns the overall potcov of a data set    "
    " Inputs: ds = AirTraf dataset routes_out    "
    "====================================================================================="""
    # Get to the potcov property
    ds_potcov = ds.isel(AirTraf_properties=9)

    # Get overall potcov by summing over all dimensions
    overall_potcov = ds_potcov.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    return overall_potcov


def get_overall_soc(ds):
    """====================================================================================="
    " Function that returns the overall Simple Operating Costs of a data set    "
    " Inputs: ds = AirTraf dataset routes_out    "
    "====================================================================================="""

    fuel_price = 1.545                  # Average fuel Price in March 2017 [US Dollar/US Gallon]
    fuel_density = 6.71                 # Fuel_density [lbs/US Gallon]
```

```python
    c_f_lbs = fuel_price/fuel_density    # Unit fuel costs [US Dollar/lbs]
    c_f_kg = c_f_lbs/0.45359237          # Unit fuel costs [US Dollar/kg]

    c_t_h = 2710                         # Unit time costs [US Dollar/h]
    c_o = 0.0                            # other costs [US Dollar]

    ds_v = ds.isel(AirTraf_properties=4, AirTraf_waypoints_out=slice(0, -1))    #
    Aircraft ground speed [km/h]
    ds_d = ds.isel(AirTraf_properties=5, AirTraf_waypoints_out=slice(1, None))  #
    Distance for each segment [km]
    ds_f = ds.isel(AirTraf_properties=6, AirTraf_waypoints_out=slice(1, None))  # Fuel
    use [kg]

    # Calculate time by dividing distance over speed
    ds_t = ds_d/ds_v

    # Calculate soc per segment in $
    ds_soc = c_t_h * ds_t + c_f_kg * ds_f + c_o

    # Get overall SOC by summing over all dimensions
    overall_soc = ds_soc.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'
    ])

    return overall_soc


def get_nighttime_segments(ds_airtraf, t_th):
    """
    ====================================================================================="

    " Function that returns whether a segment is a nighttime (1) or a daytime segment
    (0)    "
    " Inputs: ds_airtraf = AirTraf dataset, t_th = time in hours
        "

    "=====================================================================================
    """

    # ============================= PART I: preparations =============================
    # Returns the waypoint time stamps in datetime format for riseutc and setutc
    def prepare_timestamps_for_waypoints(time_ns, ref_julian_date_ns):
        # Add the ref_julian_date_ns to ds_time_ns
        julian_date = (time_ns + ref_julian_date_ns)/8.64e13
        # Turn into the datetime format
        time_stamp = pd.to_datetime(julian_date, origin='julian', unit="D")

        return time_stamp

    # Applies prepare_timestamps_for_waypoints to dataset
    def apply_prepare_timestamps_for_waypoints(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        # Get reference date (which is 1 day before first stored data)
        ref_date = ds['time'].values[0] - np.timedelta64(1, 'D')
        # [ns] Convert ref_date to Julian date for easier calculations
        ref_julian_date_ns = pd.Timestamp(ref_date).to_julian_date() * 8.64e13
        # [ns] Convert time data set to Julian date for easier calculations
        ds_time_ns = ds.isel(AirTraf_properties=3) * 8.64e13

        return xr.apply_ufunc(np.vectorize(prepare_timestamps_for_waypoints),
                              ds_time_ns,
                              ref_julian_date_ns,
                              dask='allowed')

    # Applies the sts.Suntimes function to prepare the waypoint place for riseutc and
    setutc
    def prepare_loc_for_suntimes(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        da_lon = ds.isel(AirTraf_properties=0)     # Get the longitude
        da_lat = ds.isel(AirTraf_properties=1)     # Get the latitude
        da_alt = ds.isel(AirTraf_properties=2)     # Get the altitude

        return xr.apply_ufunc(np.vectorize(sts.SunTimes), da_lon, da_lat, da_alt, dask='allowed')

    # Returns the segment time stamps in datetime format for get_is_nighttime and get_is_nighttime_segment
    def prepare_timestamps_for_segments(time_ns_0, time_ns_1, ref_julian_date_ns):
        # Find the average of the waypoints time
        average_time_ns = (time_ns_1 + time_ns_0) / 2
        # Add the ref_julian_date_ns to ds_time_ns
        average_julian_date = (average_time_ns + ref_julian_date_ns) / 8.64e13
        # Turn into the datetime format
        time_stamp = pd.to_datetime(average_julian_date, origin='julian', unit="D")

        return time_stamp

    # Applies prepare_timestamps_for_segments to dataset
    def apply_prepare_timestamps_for_segments(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        # Get reference date (which is 1 day before first stored data)
        ref_date = ds['time'].values[0] - np.timedelta64(1, 'D')
        # [ns] Convert ref_date to Julian date for easier calculations
        ref_julian_date_ns = pd.Timestamp(ref_date).to_julian_date()*8.64e13
        # [ns] Convert time data set to Julian date for easier calculations
        ds_time_ns = ds.isel(AirTraf_properties=3)*8.64e13
        # Get dataset with last waypoint removed
        ds_time_ns_0 = ds_time_ns.drop_isel(AirTraf_waypoints_out=-1)
        # Get dataset with first waypoint removed
        ds_time_ns_1 = ds_time_ns.drop_isel(AirTraf_waypoints_out=0)

        return xr.apply_ufunc(np.vectorize(prepare_timestamps_for_segments),
                              ds_time_ns_0,
                              ds_time_ns_1,
                              ref_julian_date_ns,
                              dask='allowed')

    # ===================== PART II: get time stamps of current and next-day sunrise and sunset ======================
    # Returns the current day time of sunrise with the riseutc function
    def get_time_of_sunrise(loc_prepared, timestamp_prepared, hemisphere='N'):
        date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared)/1e9)
        while True:
            try:
                time_of_sunrise = loc_prepared.riseutc(date_time_timestamp)
            # If in region of polar day or polar night
            except ValueError:
                # If in region and time of polar day
                if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                    # Search backwards for the day when the sun rose
                    date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
                # If in region and time of polar night
                else:
                    # Search forward for the day when the sun will rise
                    date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
                continue
            else:
                break
        return time_of_sunrise

    # Applies get_time_of_sunrise to dataset
    def apply_get_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared):
        return xr.apply_ufunc(np.vectorize(get_time_of_sunrise),
                              ds_loc_prepared,
                              ds_timestamp_prepared,
                              'N',
                              dask='allowed')

    # Returns the current day time of sunset with the setutc function
```

```
def get_time_of_sunset(loc_prepared, timestamp_prepared, hemisphere='N'):
    date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared) / 1e9
)
    while True:
        try:
            time_of_sunset = loc_prepared.setutc(date_time_timestamp)
        # If in region of polar day or polar night
        except ValueError:
            # If in region and time of polar day
            if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                # Search forward for the day when the sun will set
                date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
            # If in region and time of polar night
            else:
                # Search backward for the day when the sun had set
                date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
            continue
        else:
            break

    return time_of_sunset

# Applies get_time_of_sunset to dataset
def apply_get_time_of_sunset(ds_loc_prepared, ds_timestamp_prepared):
    return xr.apply_ufunc(np.vectorize(get_time_of_sunset),
                          ds_loc_prepared,
                          ds_timestamp_prepared,
                          'N',
                          dask='allowed')

# Returns the coming day time of sunrise with the riseutc function
def get_next_day_time_of_sunrise(loc_prepared, timestamp_prepared, hemisphere='N'):
    date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared) / 1e9
) + dt.timedelta(days=1)
    while True:
        try:
            next_day_time_of_sunrise = loc_prepared.riseutc(date_time_timestamp)
        # If in region of polar day or polar night
        except ValueError:
            # If in region and time of polar day
            if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                # Search backward for the day when the sun rose
                date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
            # If in region and time of polar night
            else:
                # Search forward for the day when the sun will rise
                date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
            continue
        else:
            break
    return next_day_time_of_sunrise

# Applies get_next_day_time_of_sunrise to dataset
def apply_get_next_day_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared):
    return xr.apply_ufunc(np.vectorize(get_next_day_time_of_sunrise),
                          ds_loc_prepared,
                          ds_timestamp_prepared,
                          'N',
                          dask='allowed')

# ============================ PART III: get time until sunrise and sunset
============================
# Returns the time until the next sunrise
def get_time_to_sunrise(timestamp_prepared, t_sunrise, next_day_t_sunrise):
    # If the timestamp is earlier than today's sunrise
    if np.datetime64(timestamp_prepared, 'ns') < np.datetime64(t_sunrise, 'ns'):
        time_to_sunrise = np.datetime64(t_sunrise, 'ns') - np.datetime64(
timestamp_prepared, 'ns')
    # If today's sunrise has already passed
```

```python
    else:
        time_to_sunrise = np.datetime64(next_day_t_sunrise, 'ns') - np.datetime64(timestamp_prepared, 'ns')

    return time_to_sunrise

# Applies get_time_to_sunrise to dataset
def apply_get_time_to_sunrise(ds_timestamp_prepared, ds_t_sunrise, ds_next_day_t_sunrise):
    return xr.apply_ufunc(np.vectorize(get_time_to_sunrise),
                          ds_timestamp_prepared,
                          ds_t_sunrise,
                          ds_next_day_t_sunrise,
                          dask='allowed')

# ============================ PART IV: Check if it is currently nighttime ============================
# Returns whether a waypoints is in nighttime
# Returns 1 if the current-day sunset has passed or the current-day sunrise is yet to come. Else returns 0
def get_is_night(timestamp_prepared, t_sunrise, t_sunset):
    # If the timestamp is earlier than today's sunrise OR after today's sunset
    if (np.datetime64(timestamp_prepared, 'ns') < np.datetime64(t_sunrise, 'ns')) or (np.datetime64(timestamp_prepared, 'ns') > np.datetime64(t_sunset, 'ns')):
        is_night = 1
    else:
        is_night = 0

    return is_night

# Applies get_is_night to dataset
def apply_get_is_night(ds_timestamp_segments_prepared, ds_segment_t_sunrise, ds_segment_t_sunset):
    return xr.apply_ufunc(np.vectorize(get_is_night),
                          ds_timestamp_segments_prepared,
                          ds_segment_t_sunrise,
                          ds_segment_t_sunset,
                          dask='allowed')

# ============================ PART V: Check if the segment is a nighttime segment ============================
# Returns whether a waypoint is a nighttime waypoint
# Returns 1 if it is night and the time to sunrise is larger than the threshold time to sunrise. Else returns 0
def get_is_night_segment(nighttime, time_to_sunrise, t_th, contrail_atr20):
    # If the contrail ATR20 has a value smaller than zero
    if contrail_atr20 < 0:
        is_night_segment = 0*nighttime
    # If the time until sunrise is larger than the given threshold time to sunrise
    elif np.timedelta64(time_to_sunrise, 'ns') > np.timedelta64(int(t_th * 3.6e12), 'ns'):
        is_night_segment = 1*nighttime
    else:
        is_night_segment = 0*nighttime

    return is_night_segment

# Applies  get_is_night_segment to dataset
def apply_get_is_night_segment(ds_nighttime, ds_segment_time_to_sunrise, t_th, ds_contrail_atr20):
    return xr.apply_ufunc(np.vectorize(get_is_night_segment),
                          ds_nighttime,
                          ds_segment_time_to_sunrise,
                          t_th,
                          ds_contrail_atr20,
                          dask='allowed')

# ============================ PART VI: Execute the embedded functions ============================
    # Part I
    ds_timestamp_prepared = apply_prepare_timestamps_for_waypoints(ds_airtraf)
    ds_loc_prepared = prepare_loc_for_suntimes(ds_airtraf)
    ds_timestamp_segments_prepared = apply_prepare_timestamps_for_segments(ds_airtraf)

    # Part II
    ds_t_sunrise = apply_get_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared)
    ds_t_sunset = apply_get_time_of_sunset(ds_loc_prepared, ds_timestamp_prepared)
    ds_next_day_t_sunrise = apply_get_next_day_time_of_sunrise(ds_loc_prepared,
    ds_timestamp_prepared)

    # Find the time of sunrise and sunset per segment by averaging the time of its end
    waypoints
    ds_segment_t_sunrise = ds_t_sunrise.drop_isel(AirTraf_waypoints_out=-1)+(
    ds_t_sunrise.drop_isel(AirTraf_waypoints_out=0)-ds_t_sunrise.drop_isel(
    AirTraf_waypoints_out=-1))/2
    ds_segment_t_sunset = ds_t_sunset.drop_isel(AirTraf_waypoints_out=-1)+(ds_t_sunset.
    drop_isel(AirTraf_waypoints_out=0)-ds_t_sunset.drop_isel(AirTraf_waypoints_out=-1))
    /2

    # Part III
    ds_time_to_sunrise = apply_get_time_to_sunrise(ds_timestamp_prepared, ds_t_sunrise,
    ds_next_day_t_sunrise)
    ds_segment_time_to_sunrise = (ds_time_to_sunrise.drop_isel(AirTraf_waypoints_out=-1)
    +ds_time_to_sunrise.drop_isel(AirTraf_waypoints_out=0))/2

    # Part IV
    ds_nighttime = apply_get_is_night(ds_timestamp_segments_prepared,
    ds_segment_t_sunrise, ds_segment_t_sunset)

    # Part V
    ds_nighttime_segments = apply_get_is_night_segment(ds_nighttime,
    ds_segment_time_to_sunrise, t_th, ds_airtraf["routes_out"].isel(AirTraf_properties
    =13).drop_isel(AirTraf_waypoints_out=0) * 0.9654676258992805)

    return ds_nighttime_segments


def get_average_pareto_front_waypoints(f00_airtraf_folder, f05_airtraf_folder,
    f100_airtraf_folder, t_th):
    """
    ====================================================================================="

    " Function that returns the delta specific effective contrail ATR20 and SOC
    separately  "
    " for day and night. Also a function connecting the dots is provided
        "
    " Inputs: 3x file path = 3x *airtraf_ac.nc files
        "
    "          t_th
        "

    "====================================================================================
    """
    # Open the airtraf_ac.nc files
    ds_airtraf_00 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f00_airtraf_folder))
    ds_airtraf_05 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f05_airtraf_folder))
    ds_airtraf_100 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f100_airtraf_folder))

    # Correct for 12 h time step by removing duplicates if necessary
    if ds_airtraf_00['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_00 = ds_airtraf_00.drop_isel(time=np.arange(0, len(ds_airtraf_00["
    routes_out"]), 2).tolist())
    if ds_airtraf_05['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_05 = ds_airtraf_05.drop_isel(time=np.arange(0, len(ds_airtraf_05["
    routes_out"]), 2).tolist())
    if ds_airtraf_100['time'].values[0].astype(str)[11:13] == '12':
            ds_airtraf_100 = ds_airtraf_100.drop_isel(time=np.arange(0, len(
    ds_airtraf_100["routes_out"]), 2).tolist())
```

```python

    # Get per segment contrail ATR20
    per_segment_contrail_atr20_00 = get_per_segment_contrail_atr20(ds_airtraf_00["routes_out"])
    per_segment_contrail_atr20_05 = get_per_segment_contrail_atr20(ds_airtraf_05["routes_out"])
    per_segment_contrail_atr20_100 = get_per_segment_contrail_atr20(ds_airtraf_100["routes_out"])

    # Get the per_segment SOC
    per_segment_soc_00 = get_per_segment_soc(ds_airtraf_00["routes_out"])
    per_segment_soc_05 = get_per_segment_soc(ds_airtraf_05["routes_out"])
    per_segment_soc_100 = get_per_segment_soc(ds_airtraf_100["routes_out"])

    # --------------- DISTINCT NIGHT FROM DAY --------------- #
    # Get witch segments are during daytime
    nighttime_segments_00 = get_nighttime_segments(ds_airtraf_00, t_th)
    nighttime_segments_05 = get_nighttime_segments(ds_airtraf_05, t_th)
    nighttime_segments_100 = get_nighttime_segments(ds_airtraf_100, t_th)

    # Get witch segments are during daytime
    daytime_segments_00 = (nighttime_segments_00 - 1) * -1
    daytime_segments_05 = (nighttime_segments_05 - 1) * -1
    daytime_segments_100 = (nighttime_segments_100 - 1) * -1

    # Get the night contrail ATR20 per segment
    night_per_segment_contrail_atr20_00 = per_segment_contrail_atr20_00 * nighttime_segments_00
    night_per_segment_contrail_atr20_05 = per_segment_contrail_atr20_05 * nighttime_segments_05
    night_per_segment_contrail_atr20_100 = per_segment_contrail_atr20_100 * nighttime_segments_100

    # Check if any negative nighttime contrail ATR20 occurred
    any_negative_night_contrail_atr20_00 = night_per_segment_contrail_atr20_00.where(night_per_segment_contrail_atr20_00 < 0)
    print("00 negative_night_contrail_ATR20_contributions  ", any_negative_night_contrail_atr20_00.count().values)
    any_negative_night_contrail_atr20_05 = night_per_segment_contrail_atr20_05.where(night_per_segment_contrail_atr20_05 < 0)
    print("05 negative_night_contrail_ATR20_contributions  ", any_negative_night_contrail_atr20_05.count().values)
    any_negative_night_contrail_atr20_100 = night_per_segment_contrail_atr20_100.where(night_per_segment_contrail_atr20_100 < 0)
    print("100 negative_night_contrail_ATR20_contributions ", any_negative_night_contrail_atr20_100.count().values)

    # Get the day contrail ATR20 per segment
    day_per_segment_contrail_atr20_00 = per_segment_contrail_atr20_00 * daytime_segments_00
    day_per_segment_contrail_atr20_05 = per_segment_contrail_atr20_05 * daytime_segments_05
    day_per_segment_contrail_atr20_100 = per_segment_contrail_atr20_100 * daytime_segments_100

    # Get the night SOC per segment
    night_per_segment_soc_00 = per_segment_soc_00 * nighttime_segments_00
    night_per_segment_soc_05 = per_segment_soc_05 * nighttime_segments_05
    night_per_segment_soc_100 = per_segment_soc_100 * nighttime_segments_100

    # Get the day SOC per segment
    day_per_segment_soc_00 = per_segment_soc_00 * daytime_segments_00
    day_per_segment_soc_05 = per_segment_soc_05 * daytime_segments_05
    day_per_segment_soc_100 = per_segment_soc_100 * daytime_segments_100

    # --------------- NORMALIZE W.R.T. SEGMENT EFFECTIVE KM --------------- #
    effective_distance_00 = get_effective_per_segment_km(ds_airtraf_00["routes_out"])
    effective_distance_05 = get_effective_per_segment_km(ds_airtraf_05["routes_out"])
    effective_distance_100 = get_effective_per_segment_km(ds_airtraf_100["routes_out"])

    # Get the normalized night contrail ATR20 per segment
    n_night_per_segment_contrail_atr20_00 = night_per_segment_contrail_atr20_00 /
    effective_distance_00
    n_night_per_segment_contrail_atr20_05 = night_per_segment_contrail_atr20_05 /
    effective_distance_05
    n_night_per_segment_contrail_atr20_100 = night_per_segment_contrail_atr20_100 /
    effective_distance_100

    # Get the normalized day contrail ATR20 per segment
    n_day_per_segment_contrail_atr20_00 = day_per_segment_contrail_atr20_00 /
    effective_distance_00
    n_day_per_segment_contrail_atr20_05 = day_per_segment_contrail_atr20_05 /
    effective_distance_05
    n_day_per_segment_contrail_atr20_100 = day_per_segment_contrail_atr20_100 /
    effective_distance_100

    # Get the normalized night SOC per segment
    n_night_per_segment_soc_00 = night_per_segment_soc_00 / effective_distance_00
    n_night_per_segment_soc_05 = night_per_segment_soc_05 / effective_distance_05
    n_night_per_segment_soc_100 = night_per_segment_soc_100 / effective_distance_100

    # Get the normalized day SOC per segment
    n_day_per_segment_soc_00 = day_per_segment_soc_00 / effective_distance_00
    n_day_per_segment_soc_05 = day_per_segment_soc_05 / effective_distance_05
    n_day_per_segment_soc_100 = day_per_segment_soc_100 / effective_distance_100

    # --------------- GET STANDARD DEVIATION --------------- #
    prep_std_nighttime_segments_00 = nighttime_segments_00.where(nighttime_segments_00
    != 0, np.nan)
    prep_std_nighttime_segments_05 = nighttime_segments_05.where(nighttime_segments_05
    != 0, np.nan)
    prep_std_nighttime_segments_100 = nighttime_segments_100.where(
    nighttime_segments_100 != 0, np.nan)

    prep_std_daytime_segments_00 = daytime_segments_00.where(daytime_segments_00 != 0,
    np.nan)
    prep_std_daytime_segments_05 = daytime_segments_05.where(daytime_segments_05 != 0,
    np.nan)
    prep_std_daytime_segments_100 = daytime_segments_100.where(daytime_segments_100 !=
    0, np.nan)

    prep_std_per_segment_night_contrail_atr20_00 = n_night_per_segment_contrail_atr20_00
     * prep_std_nighttime_segments_00
    prep_std_per_segment_night_contrail_atr20_05 = n_night_per_segment_contrail_atr20_05
     * prep_std_nighttime_segments_05
    prep_std_per_segment_night_contrail_atr20_100 =
    n_night_per_segment_contrail_atr20_100 * prep_std_nighttime_segments_100

    prep_std_per_segment_night_soc_00 = n_night_per_segment_soc_00 *
    prep_std_nighttime_segments_00
    prep_std_per_segment_night_soc_05 = n_night_per_segment_soc_05 *
    prep_std_nighttime_segments_05
    prep_std_per_segment_night_soc_100 = n_night_per_segment_soc_100 *
    prep_std_nighttime_segments_100

    prep_std_per_segment_day_contrail_atr20_00 = n_day_per_segment_contrail_atr20_00 *
    prep_std_daytime_segments_00
    prep_std_per_segment_day_contrail_atr20_05 = n_day_per_segment_contrail_atr20_05 *
    prep_std_daytime_segments_05
    prep_std_per_segment_day_contrail_atr20_100 = n_day_per_segment_contrail_atr20_100 *
     prep_std_daytime_segments_100

    prep_std_per_segment_day_soc_00 = n_day_per_segment_soc_00 *
    prep_std_daytime_segments_00
    prep_std_per_segment_day_soc_05 = n_day_per_segment_soc_05 *
    prep_std_daytime_segments_05
    prep_std_per_segment_day_soc_100 = n_day_per_segment_soc_100 *
    prep_std_daytime_segments_100
    prep_std_night_daily_mean_contrail_atr20_00 =
    prep_std_per_segment_night_contrail_atr20_00.mean(dim=["AirTraf_routes_out", "
    AirTraf_waypoints_out"], skipna=True)
    prep_std_night_daily_mean_contrail_atr20_05 =
    prep_std_per_segment_night_contrail_atr20_05.mean(dim=["AirTraf_routes_out", "
    AirTraf_waypoints_out"], skipna=True)
    prep_std_night_daily_mean_contrail_atr20_100 =
    prep_std_per_segment_night_contrail_atr20_100.mean(dim=["AirTraf_routes_out", "
    AirTraf_waypoints_out"], skipna=True)

    prep_std_night_daily_mean_soc_00 = prep_std_per_segment_night_soc_00.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
    prep_std_night_daily_mean_soc_05 = prep_std_per_segment_night_soc_05.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
    prep_std_night_daily_mean_soc_100 = prep_std_per_segment_night_soc_100.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)

    prep_std_day_daily_mean_contrail_atr20_00 =
    prep_std_per_segment_day_contrail_atr20_00.mean(dim=["AirTraf_routes_out", "
    AirTraf_waypoints_out"], skipna=True)
    prep_std_day_daily_mean_contrail_atr20_05 =
    prep_std_per_segment_day_contrail_atr20_05.mean(dim=["AirTraf_routes_out", "
    AirTraf_waypoints_out"], skipna=True)
    prep_std_day_daily_mean_contrail_atr20_100 =
    prep_std_per_segment_day_contrail_atr20_100.mean(dim=["AirTraf_routes_out", "
    AirTraf_waypoints_out"], skipna=True)

    prep_std_day_daily_mean_soc_00 = prep_std_per_segment_day_soc_00.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
    prep_std_day_daily_mean_soc_05 = prep_std_per_segment_day_soc_05.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
    prep_std_day_daily_mean_soc_100 = prep_std_per_segment_day_soc_100.mean(dim=["
    AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)

    prep_std_night_delta_daily_mean_contrail_atr20_05 =
    prep_std_night_daily_mean_contrail_atr20_05 -
    prep_std_night_daily_mean_contrail_atr20_00
    prep_std_night_delta_daily_mean_contrail_atr20_100 =
    prep_std_night_daily_mean_contrail_atr20_100 -
    prep_std_night_daily_mean_contrail_atr20_00

    prep_std_night_delta_daily_mean_soc_05 = prep_std_night_daily_mean_soc_05 -
    prep_std_night_daily_mean_soc_00
    prep_std_night_delta_daily_mean_soc_100 = prep_std_night_daily_mean_soc_100 -
    prep_std_night_daily_mean_soc_00

    prep_std_day_delta_daily_mean_contrail_atr20_05 =
    prep_std_day_daily_mean_contrail_atr20_05 -
    prep_std_day_daily_mean_contrail_atr20_00
    prep_std_day_delta_daily_mean_contrail_atr20_100 =
    prep_std_day_daily_mean_contrail_atr20_100 -
    prep_std_day_daily_mean_contrail_atr20_00

    prep_std_day_delta_daily_mean_soc_05 = prep_std_day_daily_mean_soc_05 -
    prep_std_day_daily_mean_soc_00
    prep_std_day_delta_daily_mean_soc_100 = prep_std_day_daily_mean_soc_100 -
    prep_std_day_daily_mean_soc_00

    std_specific_night_delta_contrail_atr20_05 =
    prep_std_night_delta_daily_mean_contrail_atr20_05.std()
    std_specific_night_delta_contrail_atr20_100 =
    prep_std_night_delta_daily_mean_contrail_atr20_100.std()

    std_specific_night_delta_soc_05 = prep_std_night_delta_daily_mean_soc_05.std()
    std_specific_night_delta_soc_100 = prep_std_night_delta_daily_mean_soc_100.std()

    std_specific_day_delta_contrail_atr20_05 =
    prep_std_day_delta_daily_mean_contrail_atr20_05.std()
    std_specific_day_delta_contrail_atr20_100 = prep_std_day_delta_daily_mean_contrail_atr20_100.std()

    std_specific_day_delta_soc_05 = prep_std_day_delta_daily_mean_soc_05.std()
    std_specific_day_delta_soc_100 = prep_std_day_delta_daily_mean_soc_100.std()

    # --------------- SUM TO A TOTAL --------------- #
    # Get total nighttime contrail ATR20
    total_nighttime_contrail_atr20_00 = n_night_per_segment_contrail_atr20_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_nighttime_contrail_atr20_05 = n_night_per_segment_contrail_atr20_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_nighttime_contrail_atr20_100 = n_night_per_segment_contrail_atr20_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get total daytime contrail ATR20
    total_daytime_contrail_atr20_00 = n_day_per_segment_contrail_atr20_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_daytime_contrail_atr20_05 = n_day_per_segment_contrail_atr20_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_daytime_contrail_atr20_100 = n_day_per_segment_contrail_atr20_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get total nighttime SOC
    total_nighttime_soc_00 = n_night_per_segment_soc_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_nighttime_soc_05 = n_night_per_segment_soc_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_nighttime_soc_100 = n_night_per_segment_soc_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get total daytime SOC
    total_daytime_soc_00 = n_day_per_segment_soc_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_daytime_soc_05 = n_day_per_segment_soc_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_daytime_soc_100 = n_day_per_segment_soc_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # --------------- NORMALIZE W.R.T. NUMBER OF SEGMENTS --------------- #
    # Count number of night segments to later calculate the mean
    n_nighttime_segments_00 = nighttime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_nighttime_segments_05 = nighttime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_nighttime_segments_100 = nighttime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    print("00 nighttime_segments  ", n_nighttime_segments_00.values)
    print("05 nighttime_segments  ", n_nighttime_segments_05.values)
    print("100 nighttime_segments ", n_nighttime_segments_100.values)

    # Count number of day segments to later calculate the mean
    n_daytime_segments_00 = daytime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_daytime_segments_05 = daytime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_daytime_segments_100 = daytime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    print("00 daytime_segments  ", n_daytime_segments_00.values)
    print("05 daytime_segments  ", n_daytime_segments_05.values)
    print("100 daytime_segments ", n_daytime_segments_100.values)

    # Get mean effective specific nighttime contrail ATR20
    mean_nighttime_contrail_atr20_00 = total_nighttime_contrail_atr20_00 / n_nighttime_segments_00
    mean_nighttime_contrail_atr20_05 = total_nighttime_contrail_atr20_05 / n_nighttime_segments_05
    mean_nighttime_contrail_atr20_100 = total_nighttime_contrail_atr20_100 / n_nighttime_segments_100

    print("00 mean_nighttime_contrail_ATR20  ", mean_nighttime_contrail_atr20_00.values, " [K/km]")
    print("05 mean_nighttime_contrail_ATR20  ", mean_nighttime_contrail_atr20_05.values, " [K/km]")
    print("100 mean_nighttime_contrail_ATR20 ", mean_nighttime_contrail_atr20_100.values, " [K/km]")

    # Get mean effective specific daytime contrail ATR20
    mean_daytime_contrail_atr20_00 = total_daytime_contrail_atr20_00 / n_daytime_segments_00
    mean_daytime_contrail_atr20_05 = total_daytime_contrail_atr20_05 / n_daytime_segments_05
    mean_daytime_contrail_atr20_100 = total_daytime_contrail_atr20_100 / n_daytime_segments_100

    print("00 mean_daytime_contrail_ATR20  ", mean_daytime_contrail_atr20_00.values, " [K/km]")
    print("05 mean_daytime_contrail_ATR20  ", mean_daytime_contrail_atr20_05.values, " [K/km]")
    print("100 mean_daytime_contrail_ATR20 ", mean_daytime_contrail_atr20_100.values, " [K/km]")

    # Get mean effective specific nighttime SOC
    mean_nighttime_soc_00 = total_nighttime_soc_00 / n_nighttime_segments_00
    mean_nighttime_soc_05 = total_nighttime_soc_05 / n_nighttime_segments_05
    mean_nighttime_soc_100 = total_nighttime_soc_100 / n_nighttime_segments_100

    print("00 mean_nighttime_SOC  ", mean_nighttime_soc_00.values, " [$/km]")
    print("05 mean_nighttime_SOC  ", mean_nighttime_soc_05.values, " [$/km]")
    print("100 mean_nighttime_SOC ", mean_nighttime_soc_100.values, " [$/km]")

    # Get mean effective specific daytime SOC
    mean_daytime_soc_00 = total_daytime_soc_00 / n_daytime_segments_00
    mean_daytime_soc_05 = total_daytime_soc_05 / n_daytime_segments_05
    mean_daytime_soc_100 = total_daytime_soc_100 / n_daytime_segments_100

    print("00 mean_daytime_SOC  ", mean_daytime_soc_00.values, " [$/km]")
    print("05 mean_daytime_SOC  ", mean_daytime_soc_05.values, " [$/km]")
    print("100 mean_daytime_SOC ", mean_daytime_soc_100.values, " [$/km]")

    # --------------- FIND CHANGE W.R.T. SOC OPTIMAL --------------- #
    # Find change in mean effective specific contrail ATR20
    delta_nighttime_contrail_atr20_00 = mean_nighttime_contrail_atr20_00 - mean_nighttime_contrail_atr20_00
    delta_nighttime_contrail_atr20_05 = mean_nighttime_contrail_atr20_05 - mean_nighttime_contrail_atr20_00
    delta_nighttime_contrail_atr20_100 = mean_nighttime_contrail_atr20_100 - mean_nighttime_contrail_atr20_00

    delta_daytime_contrail_atr20_00 = mean_daytime_contrail_atr20_00 - mean_daytime_contrail_atr20_00
    delta_daytime_contrail_atr20_05 = mean_daytime_contrail_atr20_05 - mean_daytime_contrail_atr20_00
    delta_daytime_contrail_atr20_100 = mean_daytime_contrail_atr20_100 - mean_daytime_contrail_atr20_00

    # Find change in mean effective specific SOC
    delta_nighttime_soc_00 = mean_nighttime_soc_00 - mean_nighttime_soc_00
    delta_nighttime_soc_05 = mean_nighttime_soc_05 - mean_nighttime_soc_00
    delta_nighttime_soc_100 = mean_nighttime_soc_100 - mean_nighttime_soc_00

    delta_daytime_soc_00 = mean_daytime_soc_00 - mean_daytime_soc_00
    delta_daytime_soc_05 = mean_daytime_soc_05 - mean_daytime_soc_00
    delta_daytime_soc_100 = mean_daytime_soc_100 - mean_daytime_soc_00

    # Collect in lists
```

```python
    delta_nighttime_contrail_atr20_lst = [delta_nighttime_contrail_atr20_00, delta_nighttime_contrail_atr20_05, delta_nighttime_contrail_atr20_100]
    delta_daytime_contrail_atr20_lst = [delta_daytime_contrail_atr20_00, delta_daytime_contrail_atr20_05, delta_daytime_contrail_atr20_100]

    delta_nighttime_soc_lst = [delta_nighttime_soc_00, delta_nighttime_soc_05, delta_nighttime_soc_100]
    delta_daytime_soc_lst = [delta_daytime_soc_00, delta_daytime_soc_05, delta_daytime_soc_100]

    std_plus_delta_nighttime_contrail_atr20_lst = [delta_nighttime_contrail_atr20_00,
                                                   delta_nighttime_contrail_atr20_05+std_specific_night_delta_contrail_atr20_05,
                                                   delta_nighttime_contrail_atr20_100+std_specific_night_delta_contrail_atr20_100]
    std_plus_delta_daytime_contrail_atr20_lst = [delta_daytime_contrail_atr20_00,
                                                 delta_daytime_contrail_atr20_05+std_specific_day_delta_contrail_atr20_05,
                                                 delta_daytime_contrail_atr20_100+std_specific_day_delta_contrail_atr20_100]

    std_plus_delta_nighttime_soc_lst = [delta_nighttime_soc_00,
                                        delta_nighttime_soc_05+std_specific_night_delta_soc_05,
                                        delta_nighttime_soc_100+std_specific_night_delta_soc_100]
    std_plus_delta_daytime_soc_lst = [delta_daytime_soc_00,
                                      delta_daytime_soc_05+std_specific_day_delta_soc_05,
                                      delta_daytime_soc_100+std_specific_day_delta_soc_100]

    std_min_delta_nighttime_contrail_atr20_lst = [delta_nighttime_contrail_atr20_00,
                                                  delta_nighttime_contrail_atr20_05-std_specific_night_delta_contrail_atr20_05,
                                                  delta_nighttime_contrail_atr20_100-std_specific_night_delta_contrail_atr20_100]
    std_min_delta_daytime_contrail_atr20_lst = [delta_daytime_contrail_atr20_00,
                                                delta_daytime_contrail_atr20_05-std_specific_day_delta_contrail_atr20_05,
                                                delta_daytime_contrail_atr20_100-std_specific_day_delta_contrail_atr20_100]

    std_min_delta_nighttime_soc_lst = [delta_nighttime_soc_00,
                                       delta_nighttime_soc_05-std_specific_night_delta_soc_05,
                                       delta_nighttime_soc_100-std_specific_night_delta_soc_100]
    std_min_delta_daytime_soc_lst = [delta_daytime_soc_00,
                                     delta_daytime_soc_05-std_specific_day_delta_soc_05,
                                     delta_daytime_soc_100-std_specific_day_delta_soc_100]

    # Finds straight lines to make plot more readable
    fit_nighttime = np.polyfit(delta_nighttime_contrail_atr20_lst, delta_nighttime_soc_lst, 2)
    fit_daytime = np.polyfit(delta_daytime_contrail_atr20_lst, delta_daytime_soc_lst, 2)

    func_nighttime = np.poly1d(fit_nighttime)
    func_daytime = np.poly1d(fit_daytime)

    std_plus_fit_nighttime = np.polyfit(std_plus_delta_nighttime_contrail_atr20_lst, std_plus_delta_nighttime_soc_lst, 2)
    std_plus_fit_daytime = np.polyfit(std_plus_delta_daytime_contrail_atr20_lst, std_plus_delta_daytime_soc_lst, 2)

    std_plus_func_nighttime = np.poly1d(std_plus_fit_nighttime)
    std_plus_func_daytime = np.poly1d(std_plus_fit_daytime)

    std_min_fit_nighttime = np.polyfit(std_min_delta_nighttime_contrail_atr20_lst,
    std_min_delta_nighttime_soc_lst, 2)
    std_min_fit_daytime = np.polyfit(std_min_delta_daytime_contrail_atr20_lst,
    std_min_delta_daytime_soc_lst, 2)

    std_min_func_nighttime = np.poly1d(std_min_fit_nighttime)
    std_min_func_daytime = np.poly1d(std_min_fit_daytime)

    # Calculate percentage changes for tops
    max_nighttime_relative_change_contrail_atr20 = np.round((
    delta_nighttime_contrail_atr20_100 / mean_nighttime_contrail_atr20_00 * 100).values)
    max_nighttime_relative_change_soc = np.round((delta_nighttime_soc_100 /
    mean_nighttime_soc_00 * 100).values, 1)
    max_daytime_relative_change_contrail_atr20 = np.round((
    delta_daytime_contrail_atr20_100 / mean_daytime_contrail_atr20_00 * 100).values)
    max_daytime_relative_change_soc = np.round((delta_daytime_soc_100 /
    mean_daytime_soc_00 * 100).values, 1)

    # Calculate percentage changes for ~0.5% extra SOC
    med_nighttime_relative_change_contrail_atr20 = np.round((
    delta_nighttime_contrail_atr20_05 / mean_nighttime_contrail_atr20_00 * 100).values)
    med_nighttime_relative_change_soc = np.round((delta_nighttime_soc_05 /
    mean_nighttime_soc_00 * 100).values, 1)
    med_daytime_relative_change_contrail_atr20 = np.round((
    delta_daytime_contrail_atr20_05 / mean_daytime_contrail_atr20_00 * 100).values)
    med_daytime_relative_change_soc = np.round((delta_daytime_soc_05 /
    mean_daytime_soc_00 * 100).values, 1)

    to_be_returned = (delta_daytime_contrail_atr20_lst,
                      delta_nighttime_contrail_atr20_lst,
                      delta_daytime_soc_lst,
                      delta_nighttime_soc_lst,
                      func_daytime,
                      func_nighttime,
                      max_nighttime_relative_change_contrail_atr20,
                      max_nighttime_relative_change_soc,
                      max_daytime_relative_change_contrail_atr20,
                      max_daytime_relative_change_soc,
                      nighttime_segments_00,
                      nighttime_segments_05,
                      nighttime_segments_100,
                      std_plus_delta_nighttime_contrail_atr20_lst,
                      std_plus_delta_daytime_contrail_atr20_lst,
                      std_plus_delta_nighttime_soc_lst,
                      std_plus_delta_daytime_soc_lst,
                      std_min_delta_nighttime_contrail_atr20_lst,
                      std_min_delta_daytime_contrail_atr20_lst,
                      std_min_delta_nighttime_soc_lst,
                      std_min_delta_daytime_soc_lst,
                      std_plus_func_nighttime,
                      std_plus_func_daytime,
                      std_min_func_nighttime,
                      std_min_func_daytime,
                      med_nighttime_relative_change_contrail_atr20,
                      med_nighttime_relative_change_soc,
                      med_daytime_relative_change_contrail_atr20,
                      med_daytime_relative_change_soc)

    return to_be_returned


def get_potcov_average_pareto_front_waypoints(f00_airtraf_folder, f05_airtraf_folder,
    f100_airtraf_folder, nighttime_segments_00, nighttime_segments_05,
    nighttime_segments_100):
    """
    ================================================================================"

    " Function that returns the delta specific effective potcov and SOC separately
       "
    " for day and night. Also a function connecting the dots is provided
       "
    " Inputs: 3x file path = 3x *airtraf_ac.nc files
       "

    "=====================================================================================
    """
    # Open the airtraf_ac.nc files
    ds_airtraf_00 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f00_airtraf_folder))
    ds_airtraf_05 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f05_airtraf_folder))
    ds_airtraf_100 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f100_airtraf_folder))

    # Correct for 12 h time step by removing duplicates if necessary
    if ds_airtraf_00['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_00 = ds_airtraf_00.drop_isel(time=np.arange(0, len(ds_airtraf_00["
    routes_out"]), 2).tolist())
    if ds_airtraf_05['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_05 = ds_airtraf_05.drop_isel(time=np.arange(0, len(ds_airtraf_05["
    routes_out"]), 2).tolist())
    if ds_airtraf_100['time'].values[0].astype(str)[11:13] == '12':
            ds_airtraf_100 = ds_airtraf_100.drop_isel(time=np.arange(0, len(
    ds_airtraf_100["routes_out"]), 2).tolist())

    # Get per segment potcov
    per_segment_potcov_00 = get_per_segment_potcov(ds_airtraf_00["routes_out"])
    per_segment_potcov_05 = get_per_segment_potcov(ds_airtraf_05["routes_out"])
    per_segment_potcov_100 = get_per_segment_potcov(ds_airtraf_100["routes_out"])

    # --------------- DISTINCT NIGHT FROM DAY --------------- #
    # Get witch segments are during daytime
    daytime_segments_00 = (nighttime_segments_00 - 1) * -1
    daytime_segments_05 = (nighttime_segments_05 - 1) * -1
    daytime_segments_100 = (nighttime_segments_100 - 1) * -1

    # Get the night potcov per segment
    night_per_segment_potcov_00 = per_segment_potcov_00 * nighttime_segments_00
    night_per_segment_potcov_05 = per_segment_potcov_05 * nighttime_segments_05
    night_per_segment_potcov_100 = per_segment_potcov_100 * nighttime_segments_100

    # Get the day potcov per segment
    day_per_segment_potcov_00 = per_segment_potcov_00 * daytime_segments_00
    day_per_segment_potcov_05 = per_segment_potcov_05 * daytime_segments_05
    day_per_segment_potcov_100 = per_segment_potcov_100 * daytime_segments_100

    # --------------- NORMALIZE W.R.T. SEGMENT EFFECTIVE KM --------------- #
    effective_distance_00 = get_effective_per_segment_km(ds_airtraf_00["routes_out"])
    effective_distance_05 = get_effective_per_segment_km(ds_airtraf_05["routes_out"])
    effective_distance_100 = get_effective_per_segment_km(ds_airtraf_100["routes_out"])

    # Get the normalized night potcov per segment
    n_night_per_segment_potcov_00 = night_per_segment_potcov_00 / effective_distance_00
    n_night_per_segment_potcov_05 = night_per_segment_potcov_05 / effective_distance_05
    n_night_per_segment_potcov_100 = night_per_segment_potcov_100 /
    effective_distance_100

    # Get the normalized day potcov per segment
    n_day_per_segment_potcov_00 = day_per_segment_potcov_00 / effective_distance_00
    n_day_per_segment_potcov_05 = day_per_segment_potcov_05 / effective_distance_05
    n_day_per_segment_potcov_100 = day_per_segment_potcov_100 / effective_distance_100

    # --------------- SUM TO A TOTAL --------------- #
    # Get total nighttime potcov
    total_nighttime_potcov_00 = n_night_per_segment_potcov_00.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_nighttime_potcov_05 = n_night_per_segment_potcov_05.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_nighttime_potcov_100 = n_night_per_segment_potcov_100.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

```

```python
    # Get total daytime potcov
    total_daytime_potcov_00 = n_day_per_segment_potcov_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_daytime_potcov_05 = n_day_per_segment_potcov_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    total_daytime_potcov_100 = n_day_per_segment_potcov_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # --------------- NORMALIZE W.R.T. NUMBER OF SEGMENTS --------------- #
    # Count number of night segments to later calculate the mean
    n_nighttime_segments_00 = nighttime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_nighttime_segments_05 = nighttime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_nighttime_segments_100 = nighttime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Count number of day segments to later calculate the mean
    n_daytime_segments_00 = daytime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_daytime_segments_05 = daytime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    n_daytime_segments_100 = daytime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get mean effective specific nighttime potcov
    mean_nighttime_potcov_00 = total_nighttime_potcov_00 / n_nighttime_segments_00
    mean_nighttime_potcov_05 = total_nighttime_potcov_05 / n_nighttime_segments_05
    mean_nighttime_potcov_100 = total_nighttime_potcov_100 / n_nighttime_segments_100

    print("00 mean_nighttime_potcov  ", mean_nighttime_potcov_00.values, " [km/km]")
    print("05 mean_nighttime_potcov  ", mean_nighttime_potcov_05.values, " [km/km]")
    print("100 mean_nighttime_potcov ", mean_nighttime_potcov_100.values, " [km/km]")

    # Get mean effective specific daytime potcov
    mean_daytime_potcov_00 = total_daytime_potcov_00 / n_daytime_segments_00
    mean_daytime_potcov_05 = total_daytime_potcov_05 / n_daytime_segments_05
    mean_daytime_potcov_100 = total_daytime_potcov_100 / n_daytime_segments_100

    print("00 mean_daytime_potcov  ", mean_daytime_potcov_00.values, " [km/km]")
    print("05 mean_daytime_potcov  ", mean_daytime_potcov_05.values, " [km/km]")
    print("100 mean_daytime_potcov ", mean_daytime_potcov_100.values, " [km/km]")

    return
```

```python
"""    ======================================================================================"
" File that plots the delta effective specific contrail ATR20 [K/km] as Pareto front.    "
" The day and night front are plotted separately for winter and summer. Use is made of    "
" flight data.                                                                            "
"=======================================================================================    """

# Import functions
from flight_pareto_plotter_pp_functions import *
import matplotlib.pyplot as plt


def flight_pareto_plotter_function(t_th, also_find_new_points=True, also_identify_n_best_new_points=False, n_points=1):
    print("FLIGHT DATA PARETO PLOTTER FOR t_{TH} = " + str(t_th) + "H STARTED")

    # Define paths to data
    pareto_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/Final_PP/W_' + str(t_th) + 'h/datfiles/pareto_values/'
    soc_opt_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/Final_PP/W_' + str(t_th) + 'h/f00'
    pareto_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/Final_PP/S_' + str(t_th) + 'h/datfiles/pareto_values/'
    soc_opt_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/Final_PP/S_' + str(t_th) + 'h/f00'

    # Find winter day and night delta effective specific contrail ATR20 for all solutions
    results_day_w = get_average_pareto_front_hw(soc_opt_airtraf_folder=soc_opt_airtraf_folder_w,
                                                pareto_folder=pareto_folder_w,
                                                t_th=t_th,
                                                n_min_points=1,
                                                criterium='day',
                                                allow_cooling=True)

    day_pareto_w = results_day_w[0]
    day_max_relative_change_contrail_atr20_w = results_day_w[1]
    day_max_relative_change_soc_w = results_day_w[2]

    results_night_w = get_average_pareto_front_hw(soc_opt_airtraf_folder=soc_opt_airtraf_folder_w,
                                                  pareto_folder=pareto_folder_w,
                                                  t_th=t_th,
                                                  n_min_points=1,
                                                  criterium='night',
                                                  allow_cooling=True)

    night_pareto_w = results_night_w[0]
    night_max_relative_change_contrail_atr20_w = results_night_w[1]
    night_max_relative_change_soc_w = results_night_w[2]

    # Find summer day and night delta effective specific contrail ATR20 for all solutions
    results_day_s = get_average_pareto_front_hw(soc_opt_airtraf_folder=soc_opt_airtraf_folder_s,
                                                pareto_folder=pareto_folder_s,
                                                t_th=t_th,
                                                n_min_points=1,
                                                criterium='day',
                                                allow_cooling=True)

    day_pareto_s = results_day_s[0]
    day_max_relative_change_contrail_atr20_s = results_day_s[1]
    day_max_relative_change_soc_s = results_day_s[2]

    results_night_s = get_average_pareto_front_hw(soc_opt_airtraf_folder=soc_opt_airtraf_folder_s,
                                                  pareto_folder=pareto_folder_s,
                                                  t_th=t_th,
                                                  n_min_points=1,
                                                  criterium='night',
                                                  allow_cooling=True)

    night_pareto_s = results_night_s[0]
    night_max_relative_change_contrail_atr20_s = results_night_s[1]
    night_max_relative_change_soc_s = results_night_s[2]

    # Separate contrail ATR20 from SOC
    day_delta_contrail_atr20_w = day_pareto_w[:, 1]
    day_delta_soc_w = day_pareto_w[:, 0]
    night_delta_contrail_atr20_w = night_pareto_w[:, 1]
    night_delta_soc_w = night_pareto_w[:, 0]
    day_delta_contrail_atr20_s = day_pareto_s[:, 1]
    day_delta_soc_s = day_pareto_s[:, 0]
    night_delta_contrail_atr20_s = night_pareto_s[:, 1]
    night_delta_soc_s = night_pareto_s[:, 0]

    # Plotting
    plt.rcParams.update({'font.size': 20})

    # Plot normal Pareto fronts
    fig1, (ax1) = plt.subplots(1, 1, figsize=(12, 10))

    ax1.scatter(day_delta_contrail_atr20_w, day_delta_soc_w, marker='^', color='gold',
    label='day winter')
    ax1.scatter(night_delta_contrail_atr20_w, night_delta_soc_w, marker='^', color='
    midnightblue', label='night winter')
    ax1.scatter(day_delta_contrail_atr20_s, day_delta_soc_s, marker='o', color='gold',
    label='day summer')
    ax1.scatter(night_delta_contrail_atr20_s, night_delta_soc_s, marker='o', color='
    midnightblue', label='night summer')
    ax1.set_xlabel(r'$\Delta \overline{eATR20}_{contrail}$ [K/km]')
    ax1.set_ylabel(r'$\Delta \overline{eSOC}$ [\$/km]')

    ax1.legend()
    ax1.text(min(day_delta_contrail_atr20_w), max(day_delta_soc_w), '('+str(
    day_max_relative_change_contrail_atr20_w)+'%,'+str(day_max_relative_change_soc_w)+'
    %)', horizontalalignment='left', color='goldenrod', fontsize='small')
    ax1.text(min(night_delta_contrail_atr20_w), max(night_delta_soc_w), '('+str(
    night_max_relative_change_contrail_atr20_w)+'%,'+str(night_max_relative_change_soc_w
    )+'%)', horizontalalignment='left', color='midnightblue', fontsize='small')
    ax1.text(min(day_delta_contrail_atr20_s), max(day_delta_soc_s), '('+str(
    day_max_relative_change_contrail_atr20_s)+'%,'+str(day_max_relative_change_soc_s)+'
    %)', horizontalalignment='left', color='goldenrod', fontsize='small')
    ax1.text(min(night_delta_contrail_atr20_s), max(night_delta_soc_s), '('+str(
    night_max_relative_change_contrail_atr20_s)+'%,'+str(night_max_relative_change_soc_s
    )+'%)', horizontalalignment='left', color='midnightblue', fontsize='small')

    plt.tight_layout()
    fig1.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/
    Images/plots/flight_pareto_' + str(t_th) + 'h.pdf')

    if also_find_new_points:
        new_point_results_w = new_pareto_front_finder(pareto_subset1=day_pareto_w,
                                                      pareto_subset2=night_pareto_w,
                                                      weight_factor=1)

        average_pareto_w = new_point_results_w[0]
        new_pareto_points_array_w = new_point_results_w[1]
        source_points_array1_w = new_point_results_w[2]
        source_points_array2_w = new_point_results_w[3]

        new_point_results_s = new_pareto_front_finder(pareto_subset1=day_pareto_s,
                                                      pareto_subset2=night_pareto_s,
                                                      weight_factor=1)

        average_pareto_s = new_point_results_s[0]
        new_pareto_points_array_s = new_point_results_s[1]
        source_points_array1_s = new_point_results_s[2]
        source_points_array2_s = new_point_results_s[3]

        average_delta_contrail_atr20_w = average_pareto_w[:, 1]
        average_delta_soc_w = average_pareto_w[:, 0]

        new_pareto_delta_contrail_atr20_w = new_pareto_points_array_w[:, 1]
        new_pareto_delta_soc_w = new_pareto_points_array_w[:, 0]
        source1_delta_contrail_atr20_w = source_points_array1_w[:, 1]
        source1_delta_soc_w = source_points_array1_w[:, 0]
        source2_delta_contrail_atr20_w = source_points_array2_w[:, 1]
        source2_delta_soc_w = source_points_array2_w[:, 0]

        average_delta_contrail_atr20_s = average_pareto_s[:, 1]
        average_delta_soc_s = average_pareto_s[:, 0]

        new_pareto_delta_contrail_atr20_s = new_pareto_points_array_s[:, 1]
        new_pareto_delta_soc_s = new_pareto_points_array_s[:, 0]
        source1_delta_contrail_atr20_s = source_points_array1_s[:, 1]
        source1_delta_soc_s = source_points_array1_s[:, 0]
        source2_delta_contrail_atr20_s = source_points_array2_s[:, 1]
        source2_delta_soc_s = source_points_array2_s[:, 0]

        if also_identify_n_best_new_points:
            best_new_pareto_points_w, best_source_points1_w, best_source_points2_w =
    get_best_new_pareto_points(new_pareto_points_array=new_pareto_points_array_w,

                            average_pareto=average_pareto_w,

                            source_points_array1=source_points_array1_w,

                            source_points_array2=source_points_array2_w,

                            n_points=n_points)
            best_new_pareto_points_s, best_source_points1_s, best_source_points2_s =
    get_best_new_pareto_points(new_pareto_points_array=new_pareto_points_array_s,

                            average_pareto=average_pareto_s,

                            source_points_array1=source_points_array1_s,

                            source_points_array2=source_points_array2_s,

                            n_points=n_points)

            best_new_delta_contrail_atr20_w = best_new_pareto_points_w[:, 1]
            best_new_delta_soc_w = best_new_pareto_points_w[:, 0]

            best_source1_delta_contrail_atr20_w = best_source_points1_w[:, 1]
            best_source1_delta_soc_w = best_source_points1_w[:, 0]
            best_source2_delta_contrail_atr20_w = best_source_points2_w[:, 1]
            best_source2_delta_soc_w = best_source_points2_w[:, 0]

            best_new_delta_contrail_atr20_s = best_new_pareto_points_s[:, 1]
            best_new_delta_soc_s = best_new_pareto_points_s[:, 0]

            best_source1_delta_contrail_atr20_s = best_source_points1_s[:, 1]
            best_source1_delta_soc_s = best_source_points1_s[:, 0]
            best_source2_delta_contrail_atr20_s = best_source_points2_s[:, 1]
            best_source2_delta_soc_s = best_source_points2_s[:, 0]

        if not also_identify_n_best_new_points:
            index_ex_point_w = int(len(new_pareto_points_array_w) - 100)

            ex_new_pareto_delta_contrail_atr20_w = new_pareto_delta_contrail_atr20_w[
    index_ex_point_w]
            ex_new_pareto_delta_soc_w = new_pareto_delta_soc_w[index_ex_point_w]
            ex_source1_delta_contrail_atr20_w = source1_delta_contrail_atr20_w[
    index_ex_point_w]
            ex_source1_delta_soc_w = source1_delta_soc_w[index_ex_point_w]
            ex_source2_delta_contrail_atr20_w = source2_delta_contrail_atr20_w[
    index_ex_point_w]
            ex_source2_delta_soc_w = source2_delta_soc_w[index_ex_point_w]

            index_ex_point_s = int(len(new_pareto_points_array_s)-100)

            ex_new_pareto_delta_contrail_atr20_s = new_pareto_delta_contrail_atr20_s[
    index_ex_point_s]
            ex_new_pareto_delta_soc_s = new_pareto_delta_soc_s[index_ex_point_s]
            ex_source1_delta_contrail_atr20_s = source1_delta_contrail_atr20_s[
    index_ex_point_s]
            ex_source1_delta_soc_s = source1_delta_soc_s[index_ex_point_s]
            ex_source2_delta_contrail_atr20_s = source2_delta_contrail_atr20_s[
    index_ex_point_s]
            ex_source2_delta_soc_s = source2_delta_soc_s[index_ex_point_s]

        # Plot normal Pareto fronts with newly found points winter
```

```python
        fig2, (ax2) = plt.subplots(1, 1, figsize=(12, 10))
        ax2.scatter(average_delta_contrail_atr20_w, average_delta_soc_w, marker='^', color='lightgrey', label='average')
        ax2.scatter(new_pareto_delta_contrail_atr20_w, new_pareto_delta_soc_w, marker='^', color='forestgreen', label='new')
        ax2.scatter(day_delta_contrail_atr20_w, day_delta_soc_w, marker='^', color='gold', label='day winter')
        ax2.scatter(night_delta_contrail_atr20_w, night_delta_soc_w, marker='^', color='midnightblue', label='night winter')
        if also_identify_n_best_new_points:
            ax2.scatter(best_new_delta_contrail_atr20_w, best_new_delta_soc_w, marker='^', color='fuchsia', label='best new point')
            ax2.scatter(best_source1_delta_contrail_atr20_w, best_source1_delta_soc_w, marker='^', color='darkorange', label='used day point')
            ax2.scatter(best_source2_delta_contrail_atr20_w, best_source2_delta_soc_w, marker='^', color='cyan', label='used night point')
        else:
            ax2.scatter(ex_new_pareto_delta_contrail_atr20_w, ex_new_pareto_delta_soc_w, marker='^', color='fuchsia', label='ex. new point')
            ax2.scatter(ex_source1_delta_contrail_atr20_w, ex_source1_delta_soc_w, marker='^', color='darkorange', label='ex. day point')
            ax2.scatter(ex_source2_delta_contrail_atr20_w, ex_source2_delta_soc_w, marker='^', color='cyan', label='ex. night point')
        ax2.set_xlabel(r'$\Delta \overline{eATR20}_{contrail}$ [K/km]')
        ax2.set_ylabel(r'$\Delta \overline{eSOC}$ [\$/km]')

        ax2.legend()
        plt.tight_layout()
        fig2.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/Images/plots/new_pareto_winter_' + str(t_th) + 'h.pdf')

        # Plot normal Pareto fronts with newly found points summer
        fig3, (ax3) = plt.subplots(1, 1, figsize=(12, 10))
        ax3.scatter(average_delta_contrail_atr20_s, average_delta_soc_s, marker='o', color='lightgrey', label='average')
        ax3.scatter(new_pareto_delta_contrail_atr20_s, new_pareto_delta_soc_s, marker='o', color='forestgreen', label='new')
        ax3.scatter(day_delta_contrail_atr20_s, day_delta_soc_s, marker='o', color='gold', label='day summer')
        ax3.scatter(night_delta_contrail_atr20_s, night_delta_soc_s, marker='o', color='midnightblue', label='night summer')
        if also_identify_n_best_new_points:
            ax3.scatter(best_new_delta_contrail_atr20_s, best_new_delta_soc_s, marker='o', color='fuchsia', label='best new point')
            ax3.scatter(best_source1_delta_contrail_atr20_s, best_source1_delta_soc_s, marker='o', color='darkorange', label='used day point')
            ax3.scatter(best_source2_delta_contrail_atr20_s, best_source2_delta_soc_s, marker='o', color='cyan', label='used night point')
        else:
            ax3.scatter(ex_new_pareto_delta_contrail_atr20_s, ex_new_pareto_delta_soc_s, marker='o', color='fuchsia', label='ex. new point')
            ax3.scatter(ex_source1_delta_contrail_atr20_s, ex_source1_delta_soc_s, marker='o', color='darkorange', label='ex. day point')
            ax3.scatter(ex_source2_delta_contrail_atr20_s, ex_source2_delta_soc_s, marker='o', color='cyan', label='ex.night points')
        ax3.set_xlabel(r'$\Delta \overline{eATR20}_{contrail}$ [K/km]')
        ax3.set_ylabel(r'$\Delta \overline{eSOC}$ [\$/km]')

        ax3.legend()
        plt.tight_layout()
        fig3.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/Images/plots/new_pareto_summer_' + str(t_th) + 'h.pdf')

    plt.show()

    return print("FLIGHT DATA PARETO PLOTTER FOR t_{TH} = " + str(t_th) + "H FINISHED")
```

```python
import numpy as np
import xarray as xr
import suntimes as sts
import datetime as dt
import geopy.distance
import os
from scipy.spatial.distance import cdist


def get_is_night_flight(pair_dict, key, dep_time, dep_date, ds_airtraf, t_th):
    """
    ====================================================================================="
    " Function that returns whether a flight is a daytime or nighttime flight
        "
    " Inputs: pair_dict = dictionary containing pair_distance, dep_lon, dep_lat, arr_lon
    ,   "
    "                      arr_lat as provided by get_average_pareto_front_hw()
        "
    "          key        = integer coordinate string of airport pair
        "
    "          dep_time   = string of the departure time HHMMSS
        "
    "          dep_time   = string of the departure date YYYYMMDD
        "
    "          ds_airtraf = airtraf min SOC dataset
        "

    "=====================================================================================
    """

    # Extract departure and arrival coordinates from dictionary
    dep_lon = pair_dict[key][1]
    dep_lat = pair_dict[key][2]
    arr_lon = pair_dict[key][3]
    arr_lat = pair_dict[key][4]

    # Determine approximate halfway position
    average_longitude = (dep_lon + arr_lon) / 2
    average_latitude = (dep_lat + arr_lat) / 2

    # Extract departure date from inputted departure date string
    year = dep_date[:4]
    month = dep_date[4:][:2]
    day = dep_date[6:][:2]

    # Prepare date format for suntimes functions
    suntimes_day = dt.datetime(int(year), int(month), int(day), 0, 0)

    # To find the approximate time halfway the flight
    # Find the day index of the flight
    index_year = int(year)
    index_month = int(month)
    index_day = int(day) + 1  # which is one day later than when the flight departed

    # Prepare date index string
    if index_year / 4 == int(index_year / 4):
        leap_year = True
    else:
        leap_year = False

    if index_month in [1, 3, 5, 7, 8, 10, 12]:
        n_month_days = 31
    elif index_month == 2:
        if not leap_year:
            n_month_days = 28
        else:
            n_month_days = 29
    else:
        n_month_days = 30

    if index_day > n_month_days:
        index_month = index_month + 1
        index_day = index_day - n_month_days
        if index_month > 12:
            index_year = index_year + 1
            index_month = 1

    if len(str(index_day)) < 2:
        index_day_str = '0' + str(index_day)
    else:
        index_day_str = str(index_day)

    if len(str(index_month)) < 2:
        index_month_str = '0' + str(index_month)
    else:
        index_month_str = str(index_month)

    index_year_str = str(index_year)

    index_str = index_year_str + '-' + index_month_str + '-' + index_day_str

    # Prepare next day date format for suntimes functions
    suntimes_next_day = dt.datetime(int(index_year), int(index_month), int(index_day), 0, 0)

    # Find the flight index by checking the the city pair coordinates
    for i in range(len(ds_airtraf['routes_out']['AirTraf_routes_out'])):
        ith_flight_ds = ds_airtraf['routes_out'].sel(time=index_str).isel(AirTraf_routes_out=i)
        if abs(dep_lon - ith_flight_ds.isel(AirTraf_properties=0, AirTraf_waypoints_out=0)) < 0.5:
            if abs(dep_lat - ith_flight_ds.isel(AirTraf_properties=1, AirTraf_waypoints_out=0)) < 0.5:
                if abs(arr_lon - ith_flight_ds.isel(AirTraf_properties=0, AirTraf_waypoints_out=-1)) < 0.5:
                    if abs(arr_lat - ith_flight_ds.isel(AirTraf_properties=1, AirTraf_waypoints_out=-1)) < 0.5:
                        route_index = i

    # Find the departure time
    dep_t = ds_airtraf['routes_out'].sel(time=index_str).isel(AirTraf_properties=3, AirTraf_routes_out=route_index, AirTraf_waypoints_out=0)

    # Find the arrival time
    arr_t = ds_airtraf['routes_out'].sel(time=index_str).isel(AirTraf_properties=3, AirTraf_routes_out=route_index, AirTraf_waypoints_out=-1)

    # Find the halfway time
    mid_t = (dep_t + arr_t) / 2

    # Find the time since departure in hours, minutes and seconds
    delta_t_hour = int((mid_t - dep_t) * 24)
    delta_t_minute = int((mid_t - dep_t) * 24 * 60 - delta_t_hour * 60)
    delta_t_sec = int((mid_t - dep_t) * 24 * 60 * 60 - delta_t_hour * 60 * 60 - delta_t_minute * 60)

    # Extract departure time from inputted departure time string
    dep_hour = int(dep_time[:2])
    dep_minute = int(dep_time[2:][:-2])
    dep_sec = int(dep_time[-2:])

    hour = dep_hour + delta_t_hour
    minute = dep_minute + delta_t_minute
    sec = dep_sec + delta_t_sec

    # Calculate any hour, minute of second overflow
    extra_day = 0

    if sec > 59:
        minute = minute + 1
        sec = sec - 60
    if minute > 59:
        hour = hour + 1
        minute = minute - 60
    if hour > 24:
        extra_day = 1
        hour = hour - 24

    # In datetime format, state the halfway time
    halfway_time = dt.datetime(int(year), int(month), int(day) + extra_day, hour, minute
    , sec, 0)

    # Find the departure altitude
    dep_altitude = ds_airtraf['routes_out'].sel(time=index_str).isel(AirTraf_properties
    =2, AirTraf_routes_out=route_index, AirTraf_waypoints_out=0)

    # Find the arrival altitude
    arr_altitude = ds_airtraf['routes_out'].sel(time=index_str).isel(AirTraf_properties
    =2, AirTraf_routes_out=route_index, AirTraf_waypoints_out=-1)

    average_altitude = (dep_altitude + arr_altitude) / 2

    # Find the time of utc sunrise and sunset
    sun = sts.SunTimes(longitude=average_longitude, latitude=average_latitude, altitude=
    average_altitude)
    t_rise = sun.riseutc(suntimes_day)
    t_set = sun.setutc(suntimes_day)
    t_rise_next = sun.riseutc(suntimes_next_day)

    # If in between sunrise and sunset
    if t_rise < halfway_time < t_set:
        is_nighttime = False  # It is not nighttime
    # If after sunset, but short enough until sunrise
    elif halfway_time > t_set and (t_rise_next - halfway_time) < dt.timedelta(hours=t_th
    ):
        is_nighttime = False  # It is not nighttime
    # If before sunrise, but short enough until sunrise
    elif halfway_time < t_rise and (t_rise - halfway_time) < dt.timedelta(hours=t_th):
        is_nighttime = False  # It is not nighttime
    else:
        is_nighttime = True  # else, it is nighttime

    return is_nighttime


def get_average_pareto_front_hw(soc_opt_airtraf_folder, pareto_folder, t_th,
    n_min_points=1, criterium='overall', allow_cooling=True):
    """
    ====================================================================================="

    " Function that returns the average Pareto for flights with #Pareto points>
    n_min_points "
    " Discrimination between time of day is based on approximate halfway point
        "
    " Inputs: soc_opt_airtraf_folder = folder with the *airtraf_ac.nc file(s) for
    minimum    "
    "                                  SOC
        "
    "          pareto_folder          = folder with Pareto (POBJ__.dat) files
        "
    "          n_min_pints            = minimum number of Pareto points a user wants a
        "
    "                                   front to have to be included in the calculation
        "
    "          criterium              = criterium of selection of fronts
        "
    "                                   (overall OR day OR night)
        "

```

```
"=========================================================================================
"""

# Process criterium
if criterium in ['overall', 'Overall', 'overal', 'Overal']:
    allowed_time_stamps = 'all'
elif criterium in ['day', 'Day', 'daytime', 'Daytime']:
    allowed_time_stamps = 'day'
elif criterium in ['night', 'Night', 'nighttime', 'Nighttime', 'nightime', 'Nightime
']:
    allowed_time_stamps = 'night'
else:
    print("Please check the spelling of the input 'criterium'")

# Open the airtraf_ac.nc files
soc_opt_ds_airtraf = xr.open_mfdataset('{}*airtraf_ac.nc'.format(
soc_opt_airtraf_folder))

# Correct for 12 h time step
if soc_opt_ds_airtraf['time'].values[0].astype(str)[11:13] == '12':
    soc_opt_ds_airtraf = soc_opt_ds_airtraf.drop_isel(time=np.arange(0, len(
soc_opt_ds_airtraf["routes_out"]), 2).tolist())

# Create a dictionary of the flight distances occurring in the flight plan
pair_dict = {}
total_pair_distance = 0

for i in range(len(set(soc_opt_ds_airtraf['routes_flightplan_Dtime'].values[0]))):
# For every unique airport pair
    dep_lat = soc_opt_ds_airtraf['routes_flightplan_D_lat'].values[0][i]
    dep_lon = soc_opt_ds_airtraf['routes_flightplan_D_lon'].values[0][i]
    arr_lat = soc_opt_ds_airtraf['routes_flightplan_A_lat'].values[0][i]
    arr_lon = soc_opt_ds_airtraf['routes_flightplan_A_lon'].values[0][i]
    dep_coords = (dep_lat, dep_lon)
    arr_coords = (arr_lat, arr_lon)

    dep_str = str(int(dep_lon)) + '_' + str(int(dep_lat))
    arr_str = str(int(arr_lon)) + '_' + str(int(arr_lat))
    pair_label = dep_str + '_' + arr_str  # Label consists of airport integer
coordinates

    # Calculate distance between airport pairs
    pair_distance = geopy.distance.distance(dep_coords, arr_coords).km
    total_pair_distance = total_pair_distance + pair_distance

    # Prepare for appending to dictionary
    new_dict_line = {pair_label: [pair_distance, dep_lon, dep_lat, arr_lon, arr_lat
]}

    # Append the airport pair to the dictionary
    pair_dict.update(new_dict_line)

# Prepare empty list
pareto_lst = []

# Prepare data to get a sense of relative magnitude
n_flights = 0
soc_opt_contrail_atr20_sum = 0
soc_opt_soc_sum = 0

# Open POBJ__.dat files in the specified folder and collect the data
with os.scandir(pareto_folder) as folder:
    for file in folder:
        if file.name.endswith(".dat") and file.name.startswith("POBJ") and file.
is_file():
            raw_pareto = np.loadtxt(file.path, comments="#", delimiter="        ",
unpack=False)
            is_night_flight = get_is_night_flight(pair_dict,
                                                  file.name[20:][:-4],
                                                  file.name[13:][:6],
                                                  file.name[5:][:-21],
                                                  soc_opt_ds_airtraf,
                                                  t_th)
            if is_night_flight and allowed_time_stamps == 'day':
                time_match = False
            elif not is_night_flight and allowed_time_stamps == 'night':
                time_match = False
            else:
                time_match = True

            if time_match:
                if raw_pareto.ndim == 1:  # Prevent wrong array format if POBJ__.dat
 file contains only one point
                    raw_pareto = np.array([raw_pareto])

                # Correct from pulse ATR20 to F-ATR20
                raw_pareto[:, 1] = raw_pareto[:, 1] * 0.9654676258992805

                # Remove any duplicates and sort with SOC
                sorted_pareto = np.unique(raw_pareto, axis=0)

                # Obtain the SOC-optimal SOC and contrail ATR20 for this flight
                soc_opt_soc = min(sorted_pareto[:, 0])
                soc_opt_contrail_atr20 = sorted_pareto[np.where(sorted_pareto[:, 0]
== soc_opt_soc), 1]

                if not allow_cooling:  # If cooling is not allowed
                    sorted_pareto = sorted_pareto[sorted_pareto[:, 1] >= 0]  # only
keep points with positive ATR20
                    if len(sorted_pareto) == 0:  # Correct if all points were
removed
                        sorted_pareto = np.array([[soc_opt_soc,
soc_opt_contrail_atr20]], dtype=object)

                # Calculate the change in SOC and contrail ATR20 for this flight
                delta_pareto = np.zeros(sorted_pareto.size).reshape(sorted_pareto.
shape)
                delta_pareto[:, 0] = sorted_pareto[:, 0] - soc_opt_soc
                delta_pareto[:, 1] = sorted_pareto[:, 1] - soc_opt_contrail_atr20

                this_pair_distance = pair_dict[file.name[20:][:-4]][0]

                # Normalize w.r.t. airport pair distance
                delta_pareto_per_km = delta_pareto / this_pair_distance

                # Save in Pareto list If the Pareto length is equal to or larger
than the specified minimum
                if len(delta_pareto_per_km) >= int(n_min_points):
                    # Append to the Pareto list
                    pareto_lst.append([file.name, len(delta_pareto_per_km),
delta_pareto_per_km])

                n_flights = n_flights + 1
                soc_opt_contrail_atr20_sum = soc_opt_contrail_atr20_sum +
soc_opt_contrail_atr20 / this_pair_distance
                soc_opt_soc_sum = soc_opt_soc_sum + soc_opt_soc / this_pair_distance

average_soc_opt_contrail_atr20 = soc_opt_contrail_atr20_sum / n_flights
average_soc_opt_soc = soc_opt_soc_sum / n_flights

# Convert to array
pareto_array = np.array(pareto_lst, dtype=object)

# Check which Pareto contains the larges amount of points
largest_pareto = pareto_array[np.where(pareto_array[:, 1] == max(pareto_array[:, 1])
)][0][2]

# Prepare extended Pareto array
```

```
    extended_pareto_array = pareto_array

    # extend each Pareto to match the size of the largest Pareto
    for pareto in extended_pareto_array:  # For each Pareto
        # Prepare point array matching the largest Pareto
        extended_pareto = np.zeros(largest_pareto.size).reshape(largest_pareto.shape)
        # Set point array fullness to 0
        filled = 0
        for point in pareto[2]:  # For each point
            # Find the budget of the original point
            budget = max(pareto_array[:, 1]) / pareto[1]

            if (filled - int(filled)) != 0:  # If the previous slot is not 100% filled
                rest = (int(filled + 1) - filled)  # Find rest space in previous slot
                extended_pareto[int(filled), :] = extended_pareto[int(filled), :] + rest * point  # Fill slot
                filled = filled + rest  # Update filled status
                budget = budget - rest  # Update budget status

            for i in range(int(budget)):  # Only entered if budget > 1
                extended_pareto[int(filled), :] = point  # Fill an integer number of slots fitting the budget
                filled = filled + 1  # Update filled status
                budget = budget - 1  # Update budget status

            if budget > 1e-10:  # If not all of the budget is spend
                extended_pareto[int(filled), :] = point * budget  # put rest budget in next slot
                filled = filled + budget  # Update filled status

        # Replace original Pareto by extended Pareto
        pareto[2] = extended_pareto

    # Find the average Pareto
    average_pareto = np.sum(extended_pareto_array[:, 2]) / len(extended_pareto_array)

    max_relative_change_contrail_atr20 = min(average_pareto[:, 1]) / average_soc_opt_contrail_atr20 * 100
    max_relative_change_contrail_atr20 = round(max_relative_change_contrail_atr20[0][0])
    max_relative_change_soc = round(max(average_pareto[:, 0]) / average_soc_opt_soc * 100, 1)

    to_be_returned = (average_pareto,
                      max_relative_change_contrail_atr20,
                      max_relative_change_soc)

    return to_be_returned


def new_pareto_front_finder(pareto_subset1, pareto_subset2, weight_factor=1):
    """
    ====================================================================================="
    " Function that returns a new Pareto front better than a original overall Pareto front  "
    " and the involved source points from pareto_subset1 and pareto_subset2                 "
    " Inputs: pareto_subset1 = the average Pareto point array of a subset of a Pareto data  "
    "                          set, selected by a certain criterium (e.g. day)              "
    "         pareto_subset2 = the average Pareto point array of a subset of a Pareto data  "
    "                          set, selected by a opposite criterium (e.g. night)           "
    "         weight_factor  = weight applied to pareto_subset1                             "
    "=====================================================================================
```

```python
    """

    # Resample front with cheapest mitigation limit to match other front
    if max(pareto_subset1[:, 0]) > max(pareto_subset2[:, 0]):
        resampled_contrail_atr20_2 = np.interp(pareto_subset1[:, 0], pareto_subset2[:, 0], pareto_subset2[:, 1])
        resampled_pareto_2 = np.dstack((np.where(pareto_subset1[:, 0] > max(pareto_subset2[:, 0]), max(pareto_subset2[:, 0]), pareto_subset1[:, 0]), resampled_contrail_atr20_2))
        average_pareto = np.average(np.array([weight_factor * pareto_subset1, resampled_pareto_2[0]]), axis=0)
    else:
        resampled_contrail_atr20_1 = np.interp(pareto_subset2[:, 0], pareto_subset1[:, 0], pareto_subset1[:, 1])
        resampled_pareto_1 = np.dstack((np.where(pareto_subset2[:, 0] > max(pareto_subset1[:, 0]), max(pareto_subset1[:, 0]), pareto_subset2[:, 0]), resampled_contrail_atr20_1))
        average_pareto = np.average(np.array([pareto_subset2, weight_factor * resampled_pareto_1[0]]), axis=0)

    # Prepare a list of new candidate points
    candidate_list = []
    candidate_index_list = []

    # Calculate the new candidate points
    for i in range(len(pareto_subset1)):  # For every day Pareto point
        for j in range(len(pareto_subset2)):  # For every night Pareto point
            mean_point = (weight_factor * pareto_subset1[i]+pareto_subset2[j])/2  # Find the mean
            candidate_list.append([mean_point[0], mean_point[1]])  # Store this point in candidates
            candidate_index_list.append([i, j])  # Store its indices

    # Convert to array
    candidate_array = np.array(candidate_list, dtype=object)

    # Combine with the original average Pareto front
    candidates_and_average_pareto = np.vstack((average_pareto, candidate_array))

    # Define function to find Pareto points (True) from a set of points
    # https://stackoverflow.com/questions/32791911/fast-calculation-of-pareto-front-in-python
    def is_pareto(points):
        is_efficient = np.ones(points.shape[0], dtype=bool)
        for k, c in enumerate(points):
            if is_efficient[k]:
                is_efficient[is_efficient] = np.any(points[is_efficient] < c, axis=1)  # Keep any better point
                is_efficient[k] = True  # And keep self
        return is_efficient

    # Obtain the check from the is_pareto function
    candidate_and_overall_pareto_check = is_pareto(candidates_and_average_pareto)

    # Trim off the points of the original average Pareto
    candidate_check = np.delete(candidate_and_overall_pareto_check, slice(len(average_pareto)), 0)

    # Prepare lists for the new Pareto points and their indices
    new_pareto_points_lst = []
    new_pareto_points_index_lst = []

    # Append the Pareto points if identified as is_pareto
    for p in range(len(candidate_array)):
        if candidate_check[p]:
            new_pareto_points_lst.append([candidate_array[p][0], candidate_array[p][1]])
            new_pareto_points_index_lst.append([candidate_index_list[p][0], candidate_index_list[p][1]])

    # Convert to array
    new_pareto_points_array = np.array(new_pareto_points_lst)
    new_pareto_points_index_array = np.array(new_pareto_points_index_lst)

    # Prepare list to store source points
    source_points_lst1 = []  # For pareto_subset1
    source_points_lst2 = []  # For pareto_subset2

    # Store source points
    for indices in new_pareto_points_index_array:
        index1 = indices[0]
        source_point1 = [pareto_subset1[index1][0], pareto_subset1[index1][1]]
        source_points_lst1.append(source_point1)

        index2 = indices[1]
        source_point2 = [pareto_subset2[index2][0], pareto_subset2[index2][1]]
        source_points_lst2.append(source_point2)

    # Convert to array
    source_points_array1 = np.array(source_points_lst1)
    source_points_array2 = np.array(source_points_lst2)

    to_be_returned = (average_pareto, new_pareto_points_array, source_points_array1,
    source_points_array2)

    return to_be_returned


def get_best_new_pareto_points(new_pareto_points_array, average_pareto,
    source_points_array1, source_points_array2, n_points=5):
    """
    ====================================================================================="

    " Function that returns the best Pareto points of a new Pareto front and the
    involved   "
    " source points
        "
    " Inputs: new_pareto_points_array = array of the new Pareto points
        "
    "          overall_pareto = the overall average Pareto point array of a Pareto data
    set  "
    "          source_points_array1 = involved source points from pareto_subset1
        "
    "          source_points_array2 = involved source points from pareto_subset2
        "
    "          n_points = the amount of best points
        "

    "=====================================================================================
    """

    # Sample the polynomial fit
    sampled_new_points = np.interp(np.linspace(min(average_pareto[:, 1]), 0, 5000),
    average_pareto[:, 1], average_pareto[:, 0])

    # Combine the atr20 values and soc values into an array
    atr_values = [[item] for item in np.linspace(min(average_pareto[:, 1]), 0, 5000)]
    soc_values = [[item] for item in sampled_new_points]
    sampled_new_points = np.hstack((soc_values, atr_values))

    # Prepare a list of the minimum distance for new points to the sampled overall
    points
    minimum_distance_lst = []

    # Find the minimum distance for new points to the sampled overall points
    for point in new_pareto_points_array:  # For each new Pareto point
        # Get the distance to every overall Pareto point
        distances_to_average_pareto_atr20 = cdist(np.array([point]), sampled_new_points)
        # Get the index of the minimum of those distances
        minimum_distance_index = np.argmin(distances_to_average_pareto_atr20)
        # Get the minimum of those distances
        minimum_distance = distances_to_average_pareto_atr20[0][minimum_distance_index]
        # Append the distance to the minimum_distance_lst
        minimum_distance_lst.append(minimum_distance)

    # Covert list to array
    minimum_distance_array = np.array(minimum_distance_lst)

    # Find the index of the n points with the largest distance to the overall Pareto
    best_point_index_array = np.argpartition(minimum_distance_array, -n_points)[-
    n_points:]

    # Save the n points with the largest distance to the overall Pareto and their source
     points
    best_new_pareto_points = new_pareto_points_array[best_point_index_array]
    best_source_points1 = source_points_array1[best_point_index_array]
    best_source_points2 = source_points_array2[best_point_index_array]

    return best_new_pareto_points, best_source_points1, best_source_points2
```

```
"""
    ======================================================================================"
" File that plots the delta effective specific contrail ATR20 [K/km] as Pareto front
    "
" with excluded extra cooling.bThe day and night front are plotted separately for
    "
" winter and summer. Use is made of waypoint data.
    "
"=====================================================================================
    """

# Import functions
from waypoint_pareto_plotter_no_extra_cooling_pp_functions import *
import matplotlib.pyplot as plt


def waypoint_pareto_plotter_no_extra_cooling_function(t_th):
    print("#*#*# WAYPOINT DATA PARETO PLOTTER NO EXTRA COOLING FOR t_{TH} = " + str(t_th
    ) + "H STARTED #*#*#")

    # --------------- GET TO DATA ---------------- #
    # Define paths to data
    f00_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/W_' + str(t_th) + 'h/f00'
    f05_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/W_' + str(t_th) + 'h/f05'
    f100_airtraf_folder_w = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes
    /Final_PP/W_' + str(t_th) + 'h/f100'
    f00_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/S_' + str(t_th) + 'h/f00'
    f05_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes/
    Final_PP/S_' + str(t_th) + 'h/f05'
    f100_airtraf_folder_s = 'C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Codes
    /Final_PP/S_' + str(t_th) + 'h/f100'

    print("[[[[[[[[[[[[[[[[[[[[[[[[WINTER]]]]]]]]]]]]]]]]]]]]]]]")
    results_w = get_average_pareto_front_waypoints_no_extra_cooling(f00_airtraf_folder=
    f00_airtraf_folder_w,
                                                                    f05_airtraf_folder=
    f05_airtraf_folder_w,
                                                                    f100_airtraf_folder=
    f100_airtraf_folder_w,
                                                                    t_th=t_th)

    delta_daytime_contrail_atr20_lst_w = results_w[0]
    delta_nighttime_contrail_atr20_lst_w = results_w[1]
    delta_daytime_soc_lst_w = results_w[2]
    delta_nighttime_soc_lst_w = results_w[3]
    func_daytime_w = results_w[4]
    func_nighttime_w = results_w[5]
    max_nighttime_relative_change_contrail_atr20_w = results_w[6]
    max_nighttime_relative_change_soc_w = results_w[7]
    max_daytime_relative_change_contrail_atr20_w = results_w[8]
    max_daytime_relative_change_soc_w = results_w[9]
    nighttime_segments_00_w = results_w[10]
    treated_nighttime_segments_05_w = results_w[11]
    treated_nighttime_segments_100_w = results_w[12]
    treated_std_plus_delta_nighttime_contrail_atr20_lst_w = results_w[13]
    treated_std_plus_delta_daytime_contrail_atr20_lst_w = results_w[14]
    treated_std_plus_delta_nighttime_soc_lst_w = results_w[15]
    treated_std_plus_delta_daytime_soc_lst_w = results_w[16]
    treated_std_min_delta_nighttime_contrail_atr20_lst_w = results_w[17]
    treated_std_min_delta_daytime_contrail_atr20_lst_w = results_w[18]
    treated_std_min_delta_nighttime_soc_lst_w = results_w[19]
    treated_std_min_delta_daytime_soc_lst_w = results_w[20]
    treated_std_plus_func_nighttime_w = results_w[21]
    treated_std_plus_func_daytime_w = results_w[22]
    treated_std_min_func_nighttime_w = results_w[23]
    treated_std_min_func_daytime_w = results_w[24]
    med_nighttime_relative_change_contrail_atr20_w = results_w[25]
    med_nighttime_relative_change_soc_w = results_w[26]
    med_daytime_relative_change_contrail_atr20_w = results_w[27]
    med_daytime_relative_change_soc_w = results_w[28]

    get_potcov_average_pareto_front_waypoints_no_extra_cooling(f00_airtraf_folder=
    f00_airtraf_folder_w,
                                                               f05_airtraf_folder=
    f05_airtraf_folder_w,
                                                               f100_airtraf_folder=
    f100_airtraf_folder_w,
                                                               nighttime_segments_00=
    nighttime_segments_00_w,

    treated_nighttime_segments_05=treated_nighttime_segments_05_w,

    treated_nighttime_segments_100=treated_nighttime_segments_100_w)

    print("[[[[[[[[[[[[[[[[[[[[[[[[SUMMER]]]]]]]]]]]]]]]]]]]]]]]]")
    results_s = get_average_pareto_front_waypoints_no_extra_cooling(f00_airtraf_folder=
    f00_airtraf_folder_s,
                                                                    f05_airtraf_folder=
    f05_airtraf_folder_s,
                                                                    f100_airtraf_folder=
    f100_airtraf_folder_s,
                                                                    t_th=t_th)

    delta_daytime_contrail_atr20_lst_s = results_s[0]
    delta_nighttime_contrail_atr20_lst_s = results_s[1]
    delta_daytime_soc_lst_s = results_s[2]
    delta_nighttime_soc_lst_s = results_s[3]
    func_daytime_s = results_s[4]
    func_nighttime_s = results_s[5]
    max_nighttime_relative_change_contrail_atr20_s = results_s[6]
    max_nighttime_relative_change_soc_s = results_s[7]
    max_daytime_relative_change_contrail_atr20_s = results_s[8]
    max_daytime_relative_change_soc_s = results_s[9]
    nighttime_segments_00_s = results_s[10]
    treated_nighttime_segments_05_s = results_s[11]
    treated_nighttime_segments_100_s = results_s[12]
    treated_std_plus_delta_nighttime_contrail_atr20_lst_s = results_s[13]
    treated_std_plus_delta_daytime_contrail_atr20_lst_s = results_s[14]
    treated_std_plus_delta_nighttime_soc_lst_s = results_s[15]
    treated_std_plus_delta_daytime_soc_lst_s = results_s[16]
    treated_std_min_delta_nighttime_contrail_atr20_lst_s = results_s[17]
    treated_std_min_delta_daytime_contrail_atr20_lst_s = results_s[18]
    treated_std_min_delta_nighttime_soc_lst_s = results_s[19]
    treated_std_min_delta_daytime_soc_lst_s = results_s[20]
    treated_std_plus_func_nighttime_s = results_s[21]
    treated_std_plus_func_daytime_s = results_s[22]
    treated_std_min_func_nighttime_s = results_s[23]
    treated_std_min_func_daytime_s = results_s[24]
    med_nighttime_relative_change_contrail_atr20_s = results_s[25]
    med_nighttime_relative_change_soc_s = results_s[26]
    med_daytime_relative_change_contrail_atr20_s = results_s[27]
    med_daytime_relative_change_soc_s = results_s[28]

    get_potcov_average_pareto_front_waypoints_no_extra_cooling(f00_airtraf_folder=f00_airtraf_folder_s,
                                                               f05_airtraf_folder=f05_airtraf_folder_s,
                                                               f100_airtraf_folder=f100_airtraf_folder_s,
                                                               nighttime_segments_00=nighttime_segments_00_s,
                                                               treated_nighttime_segments_05=treated_nighttime_segments_05_s,
                                                               treated_nighttime_segments_100=treated_nighttime_segments_100_s)

    # Plotting
    plt.rcParams.update({'font.size': 20})
    fig, (ax) = plt.subplots(1, 1, figsize=(12, 10))

    ax.scatter(delta_daytime_contrail_atr20_lst_w, delta_daytime_soc_lst_w, marker='^', color='gold', label='day winter')
    ax.scatter(delta_nighttime_contrail_atr20_lst_w, delta_nighttime_soc_lst_w, marker='^', color='midnightblue', label='night winter')
    ax.plot(delta_nighttime_contrail_atr20_lst_w, func_nighttime_w(delta_nighttime_contrail_atr20_lst_w), color='midnightblue', linestyle='dashed')
    ax.plot(delta_daytime_contrail_atr20_lst_w, func_daytime_w(delta_daytime_contrail_atr20_lst_w), color='gold', linestyle='dashed')
    ax.fill(np.append(treated_std_min_delta_nighttime_contrail_atr20_lst_w, treated_std_plus_delta_nighttime_contrail_atr20_lst_w[::-1]),
            np.append(delta_nighttime_soc_lst_w, delta_nighttime_soc_lst_w[::-1]),
            color='midnightblue',
            alpha=0.1)
    ax.fill(np.append(treated_std_min_delta_daytime_contrail_atr20_lst_w, treated_std_plus_delta_daytime_contrail_atr20_lst_w[::-1]),
            np.append(delta_daytime_soc_lst_w, delta_daytime_soc_lst_w[::-1]),
            color='gold',
            alpha=0.1)

    ax.scatter(delta_daytime_contrail_atr20_lst_s, delta_daytime_soc_lst_s, marker='o', color='gold', label='day summer')
    ax.scatter(delta_nighttime_contrail_atr20_lst_s, delta_nighttime_soc_lst_s, marker='o', color='midnightblue', label='night summer')
    ax.plot(delta_nighttime_contrail_atr20_lst_s, func_nighttime_s(delta_nighttime_contrail_atr20_lst_s), color='midnightblue', linestyle='dashed')
    ax.plot(delta_daytime_contrail_atr20_lst_s, func_daytime_s(delta_daytime_contrail_atr20_lst_s), color='gold', linestyle='dashed')
    ax.fill(np.append(treated_std_min_delta_nighttime_contrail_atr20_lst_s, treated_std_plus_delta_nighttime_contrail_atr20_lst_s[::-1]),
            np.append(delta_nighttime_soc_lst_s, delta_nighttime_soc_lst_s[::-1]),
            color='midnightblue',
            alpha=0.1)
    ax.fill(np.append(treated_std_min_delta_daytime_contrail_atr20_lst_s, treated_std_plus_delta_daytime_contrail_atr20_lst_s[::-1]),
            np.append(delta_daytime_soc_lst_s, delta_daytime_soc_lst_s[::-1]),
            color='gold',
            alpha=0.1)

    ax.set_xlabel(r'$\Delta \overline{eATR20}_{contrail}$ [K/km]')
    ax.set_ylabel(r'$\Delta \overline{eSOC}$ [\$/km]')
    ax.legend()

    ax.text(min(delta_daytime_contrail_atr20_lst_w), max(delta_daytime_soc_lst_w), '('+
    str(max_daytime_relative_change_contrail_atr20_w)+'%,'+str(
    max_daytime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(min(delta_nighttime_contrail_atr20_lst_w), max(delta_nighttime_soc_lst_w), '
    ('+str(max_nighttime_relative_change_contrail_atr20_w)+'%,'+str(
    max_nighttime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')
    ax.text(min(delta_daytime_contrail_atr20_lst_s), max(delta_daytime_soc_lst_s), '('+
    str(max_daytime_relative_change_contrail_atr20_s)+'%,'+str(
    max_daytime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(min(delta_nighttime_contrail_atr20_lst_s), max(delta_nighttime_soc_lst_s), '
    ('+str(max_nighttime_relative_change_contrail_atr20_s)+'%,'+str(
    max_nighttime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')

    ax.text(delta_daytime_contrail_atr20_lst_w[1], delta_daytime_soc_lst_w[1], '('+str(
    med_daytime_relative_change_contrail_atr20_w)+'%,'+str(
    med_daytime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(delta_nighttime_contrail_atr20_lst_w[1], delta_nighttime_soc_lst_w[1], '('+
    str(med_nighttime_relative_change_contrail_atr20_w)+'%,'+str(
    med_nighttime_relative_change_soc_w)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')
    ax.text(delta_daytime_contrail_atr20_lst_s[1], delta_daytime_soc_lst_s[1], '('+str(
    med_daytime_relative_change_contrail_atr20_s)+'%,'+str(
    med_daytime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    goldenrod', fontsize='small')
    ax.text(delta_nighttime_contrail_atr20_lst_s[1], delta_nighttime_soc_lst_s[1], '('+
    str(med_nighttime_relative_change_contrail_atr20_s)+'%,'+str(
    med_nighttime_relative_change_soc_s)+'%)', horizontalalignment='left', color='
    midnightblue', fontsize='small')

    plt.tight_layout()
    fig.savefig('C:/Users/wesse/Documents/AE/Master/Quarter 2.1/Thesis/Main Phase/Images
    /plots/waypoint_pareto_no_cooling_' + str(t_th) + 'h.pdf')

    plt.show()

    return print("#*#*# WAYPOINT DATA PARETO PLOTTER NO EXTRA COOLING FOR t_{TH} = " +
    str(t_th) + "H FINISHED #*#*#")
```

```
import numpy as np
import xarray as xr
import suntimes as sts
import datetime as dt
import pandas as pd


def get_per_segment_km(ds):
    """
    ==========================================================================================
    " Function that returns the per segment km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "

    "==========================================================================================
    """
    # Get to the segment kilometer property
    ds_km = ds.isel(AirTraf_properties=5).drop_isel(AirTraf_waypoints_out=0)

    return ds_km


def get_effective_per_segment_km(ds):
    """
    ==========================================================================================
    " Function that returns the  effective per segment km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "=====================================================================================
    """
    # Find the jump in longitude and latitude for each segment
    segment_lon_1 = ds.isel(AirTraf_properties=0).drop_isel(AirTraf_waypoints_out=-1)
    segment_lon_2 = ds.isel(AirTraf_properties=0).drop_isel(AirTraf_waypoints_out=0)
    segment_delta_lon = segment_lon_2 - segment_lon_1
    segment_lat_1 = ds.isel(AirTraf_properties=1).drop_isel(AirTraf_waypoints_out=-1)
    segment_lat_2 = ds.isel(AirTraf_properties=1).drop_isel(AirTraf_waypoints_out=0)
    segment_delta_lat = segment_lat_2 - segment_lat_1

    # prevent later division by zero
    segment_delta_lon_safe = segment_delta_lon.where(abs(segment_delta_lon) > 0.00001,
    0.00001)

    # Find the jump in longitude and latitude for each flight
    flight_lon_1 = ds.isel(AirTraf_properties=0).isel(AirTraf_waypoints_out=0)
    flight_lon_2 = ds.isel(AirTraf_properties=0).isel(AirTraf_waypoints_out=-1)
    flight_delta_lon = flight_lon_2 - flight_lon_1
    flight_lat_1 = ds.isel(AirTraf_properties=1).isel(AirTraf_waypoints_out=0)
    flight_lat_2 = ds.isel(AirTraf_properties=1).isel(AirTraf_waypoints_out=-1)
    flight_delta_lat = flight_lat_2 - flight_lat_1

    # prevent later division by zero
    flight_delta_lon_safe = flight_delta_lon.where(abs(flight_delta_lon) > 0.00001,
    0.00001)

    # Take the arctan of the jump in latitude over the jump in longitude to get the
    angle
    segment_orientation = np.arctan(segment_delta_lat / segment_delta_lon_safe)
    flight_orientation = np.arctan(flight_delta_lat / flight_delta_lon_safe)

    # Find the difference in angle
    delta_orientation = abs(segment_orientation - flight_orientation)

    # Find the corresponding cosine
    segment_cos = np.cos(delta_orientation)

    # Get the plain distance
    per_segment_km = get_per_segment_km(ds)

    # Compute the effective component of the plain distance
    effective_distance = segment_cos * per_segment_km

    return effective_distance


def get_per_segment_contrail_atr20(ds):
    """
    ====================================================================================="

    " Function that returns the per segment contrail ATR20 of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "=====================================================================================
    """
    # Get to the contrail atr20 property and correct from pulse ATR20 to F-ATR20
    ds_contrail_atr20 = ds.isel(AirTraf_properties=13).drop_isel(AirTraf_waypoints_out
    =0)*0.9654676258992805

    return ds_contrail_atr20

```

```
def get_per_segment_potcov(ds):
    """
    ====================================================================================="
    " Function that returns the per segment potcov of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "=====================================================================================
    """
    # Get to the potcov property
    ds_potcov = ds.isel(AirTraf_properties=9).drop_isel(AirTraf_waypoints_out=0)

    return ds_potcov


def get_per_segment_soc(ds):
    """
    ====================================================================================="
    " Function that returns the per segment Simple Operating Costs of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "=====================================================================================
    """
    fuel_price = 1.545                      # Average fuel Price in March 2017 [US Dollar/
    US Gallon]
    fuel_density = 6.71                     # Fuel_density [lbs/US Gallon]
    c_f_lbs = fuel_price/fuel_density       # Unit fuel costs [US Dollar/lbs]
    c_f_kg = c_f_lbs/0.45359237             # Unit fuel costs [US Dollar/kg]

    c_t_h = 2710                            # Unit time costs [US Dollar/h]
    c_o = 0.0                               # other costs [US Dollar]

    ds_v = ds.isel(AirTraf_properties=4, AirTraf_waypoints_out=slice(0, -1))    #
    Aircraft ground speed [km/h]
    ds_d = ds.isel(AirTraf_properties=5, AirTraf_waypoints_out=slice(1, None))  #
    Distance for each segment [km]
    ds_f = ds.isel(AirTraf_properties=6, AirTraf_waypoints_out=slice(1, None))  # Fuel
    use [kg]

    # Calculate time by dividing distance over speed
    ds_t = ds_d/ds_v

    # Calculate soc per segment in $
    ds_soc = c_t_h * ds_t + c_f_kg * ds_f + c_o

    return ds_soc


def get_overall_km(ds):
    """
    ====================================================================================="
    " Function that returns the overall km of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "=====================================================================================
    """
    # Get to the contrail atr20 property
    ds_km = ds.isel(AirTraf_properties=5)

    # Get overall contrail atr20 by summing over all dimensions
    overall_km = ds_km.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    return overall_km


def get_overall_contrail_atr20(ds):
    """
    ==================================================================================="
    " Function that returns the overall contrail ATR20 of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "===================================================================================
    """
    # Get to the contrail atr20 property and correct from pulse ATR20 to F-ATR20
    ds_contrail_atr20 = ds.isel(AirTraf_properties=13)*0.9654676258992805

    # Get overall contrail atr20 by summing over all dimensions
    overall_contrail_atr20 = ds_contrail_atr20.sum(dim=['AirTraf_waypoints_out', '
    AirTraf_routes_out', 'time'])

    return overall_contrail_atr20


def get_overall_potcov(ds):
    """
    ==================================================================================="
    " Function that returns the overall potcov of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "===================================================================================
    """
    # Get to the potcov property
    ds_potcov = ds.isel(AirTraf_properties=9)

    # Get overall potcov by summing over all dimensions
    overall_potcov = ds_potcov.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', '
    time'])

    return overall_potcov


def get_overall_soc(ds):
    """
    ==================================================================================="
    " Function that returns the overall Simple Operating Costs of a data set
        "
    " Inputs: ds = AirTraf dataset routes_out
        "
    "===================================================================================
    """
    fuel_price = 1.545                     # Average fuel Price in March 2017 [US Dollar/
    US Gallon]
    fuel_density = 6.71                    # Fuel_density [lbs/US Gallon]
    c_f_lbs = fuel_price/fuel_density      # Unit fuel costs [US Dollar/lbs]
    c_f_kg = c_f_lbs/0.45359237            # Unit fuel costs [US Dollar/kg]

    c_t_h = 2710                           # Unit time costs [US Dollar/h]
    c_o = 0.0                              # other costs [US Dollar]

    ds_v = ds.isel(AirTraf_properties=4, AirTraf_waypoints_out=slice(0, -1))    #
    Aircraft ground speed [km/h]
    ds_d = ds.isel(AirTraf_properties=5, AirTraf_waypoints_out=slice(1, None))  #
    Distance for each segment [km]
    ds_f = ds.isel(AirTraf_properties=6, AirTraf_waypoints_out=slice(1, None))  # Fuel
    use [kg]

    # Calculate time by dividing distance over speed
    ds_t = ds_d/ds_v

    # Calculate soc per segment in $
    ds_soc = c_t_h * ds_t + c_f_kg * ds_f + c_o

    # Get overall SOC by summing over all dimensions
    overall_soc = ds_soc.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'
    ])

    return overall_soc


def get_nighttime_segments(ds_airtraf, t_th):
    """
    ==================================================================================="

    " Function that returns whether a segment is a nighttime (1) or a daytime segment
    (0)   "
    " Inputs: ds_airtraf = AirTraf dataset, t_th = time in hours
        "

    "===================================================================================
    """

    # ============================ PART I: preparations ============================
    # Returns the waypoint time stamps in datetime format for riseutc and setutc
    def prepare_timestamps_for_waypoints(time_ns, ref_julian_date_ns):
        # Add the ref_julian_date_ns to ds_time_ns
        julian_date = (time_ns + ref_julian_date_ns) / 8.64e13
        # Turn into the datetime format
        time_stamp = pd.to_datetime(julian_date, origin='julian', unit="D")

        return time_stamp

    # Applies prepare_timestamps_for_waypoints to dataset
    def apply_prepare_timestamps_for_waypoints(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        # Get reference date (which is 1 day before first stored data)
        ref_date = ds['time'].values[0] - np.timedelta64(1, 'D')
        # [ns] Convert ref_date to Julian date for easier calculations
        ref_julian_date_ns = pd.Timestamp(ref_date).to_julian_date() * 8.64e13
        # [ns] Convert time data set to Julian date for easier calculations
        ds_time_ns = ds.isel(AirTraf_properties=3) * 8.64e13

        return xr.apply_ufunc(np.vectorize(prepare_timestamps_for_waypoints),
                              ds_time_ns,
                              ref_julian_date_ns,
                              dask='allowed')

    # Applies the sts.Suntimes function to prepare the waypoint place for riseutc and
    setutc
    def prepare_loc_for_suntimes(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        da_lon = ds.isel(AirTraf_properties=0)     # Get the longitude
        da_lat = ds.isel(AirTraf_properties=1)     # Get the latitude
        da_alt = ds.isel(AirTraf_properties=2)     # Get the altitude

        return xr.apply_ufunc(np.vectorize(sts.SunTimes), da_lon, da_lat, da_alt, dask='
    allowed')

    # Returns the segment time stamps in datetime format for get_is_nighttime and
    get_is_nighttime_segment
    def prepare_timestamps_for_segments(time_ns_0, time_ns_1, ref_julian_date_ns):
        # Find the average of the waypoints time
        average_time_ns = (time_ns_1 + time_ns_0) / 2
        # Add the ref_julian_date_ns to ds_time_ns
        average_julian_date = (average_time_ns + ref_julian_date_ns) / 8.64e13
        # Turn into the datetime format
        time_stamp = pd.to_datetime(average_julian_date, origin='julian', unit="D")

        return time_stamp

    # Applies prepare_timestamps_for_segments to dataset
    def apply_prepare_timestamps_for_segments(ds_airtraf):
        ds = ds_airtraf["routes_out"]
        # Get reference date (which is 1 day before first stored data)
        ref_date = ds['time'].values[0] - np.timedelta64(1, 'D')
        # [ns] Convert ref_date to Julian date for easier calculations
        ref_julian_date_ns = pd.Timestamp(ref_date).to_julian_date() * 8.64e13
        # [ns] Convert time data set to Julian date for easier calculations
        ds_time_ns = ds.isel(AirTraf_properties=3) * 8.64e13
        # Get dataset with last waypoint removed
        ds_time_ns_0 = ds_time_ns.drop_isel(AirTraf_waypoints_out=-1)
        # Get dataset with first waypoint removed
        ds_time_ns_1 = ds_time_ns.drop_isel(AirTraf_waypoints_out=0)

        return xr.apply_ufunc(np.vectorize(prepare_timestamps_for_segments),
                              ds_time_ns_0,
                              ds_time_ns_1,
                              ref_julian_date_ns,
                              dask='allowed')

    # ===================== PART II: get time stamps of current and next-day sunrise
    and sunset =====================
    # Returns the current day time of sunrise with the riseutc function
    def get_time_of_sunrise(loc_prepared, timestamp_prepared, hemisphere='N'):
        date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared)/1e9)
        while True:
            try:
                time_of_sunrise = loc_prepared.riseutc(date_time_timestamp)
            # If in region of polar day or polar night
            except ValueError:
                # If in region and time of polar day
                if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                    # Search backwards for the day when the sun rose
                    date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
                # If in region and time of polar night
                else:
                    # Search forward for the day when the sun will rise
                    date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
                continue
            else:
                break
        return time_of_sunrise

    # Applies get_time_of_sunrise to dataset
    def apply_get_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared):
        return xr.apply_ufunc(np.vectorize(get_time_of_sunrise),
                              ds_loc_prepared,
                              ds_timestamp_prepared,
                              'N',
                              dask='allowed')

    # Returns the current day time of sunset with the setutc function
    def get_time_of_sunset(loc_prepared, timestamp_prepared, hemisphere='N'):
        date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared) / 1e9
    )
        while True:
            try:
                time_of_sunset = loc_prepared.setutc(date_time_timestamp)
            # If in region of polar day or polar night
            except ValueError:
                # If in region and time of polar day
```

```python
            if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                # Search forward for the day when the sun will set
                date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
            # If in region and time of polar night
            else:
                # Search backward for the day when the sun had set
                date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
            continue
        else:
            break

    return time_of_sunset

# Applies get_time_of_sunset to dataset
def apply_get_time_of_sunset(ds_loc_prepared, ds_timestamp_prepared):
    return xr.apply_ufunc(np.vectorize(get_time_of_sunset),
                          ds_loc_prepared,
                          ds_timestamp_prepared,
                          'N',
                          dask='allowed')

# Returns the coming day time of sunrise with the riseutc function
def get_next_day_time_of_sunrise(loc_prepared, timestamp_prepared, hemisphere='N'):
    date_time_timestamp = dt.datetime.utcfromtimestamp(int(timestamp_prepared) / 1e9
) + dt.timedelta(days=1)
    while True:
        try:
            next_day_time_of_sunrise = loc_prepared.riseutc(date_time_timestamp)
        # If in region of polar day or polar night
        except ValueError:
            # If in region and time of polar day
            if hemisphere == 'N' and 2 < date_time_timestamp.month < 10:
                # Search backward for the day when the sun rose
                date_time_timestamp = date_time_timestamp - dt.timedelta(days=1)
            # If in region and time of polar night
            else:
                # Search forward for the day when the sun will rise
                date_time_timestamp = date_time_timestamp + dt.timedelta(days=1)
            continue
        else:
            break
    return next_day_time_of_sunrise

# Applies get_next_day_time_of_sunrise to dataset
def apply_get_next_day_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared):
    return xr.apply_ufunc(np.vectorize(get_next_day_time_of_sunrise),
                          ds_loc_prepared,
                          ds_timestamp_prepared,
                          'N',
                          dask='allowed')

# ============================ PART III: get time until sunrise and sunset
============================
# Returns the time until the next sunrise
def get_time_to_sunrise(timestamp_prepared, t_sunrise, next_day_t_sunrise):
    # If the timestamp is earlier than today's sunrise
    if np.datetime64(timestamp_prepared, 'ns') < np.datetime64(t_sunrise, 'ns'):
        time_to_sunrise = np.datetime64(t_sunrise, 'ns') - np.datetime64(
timestamp_prepared, 'ns')
    # If today's sunrise has already passed
    else:
        time_to_sunrise = np.datetime64(next_day_t_sunrise, 'ns') - np.datetime64(
timestamp_prepared, 'ns')

    return time_to_sunrise

# Applies get_time_to_sunrise to dataset
def apply_get_time_to_sunrise(ds_timestamp_prepared, ds_t_sunrise,
ds_next_day_t_sunrise):
    return xr.apply_ufunc(np.vectorize(get_time_to_sunrise),
                          ds_timestamp_prepared,
                          ds_t_sunrise,
                          ds_next_day_t_sunrise,
                          dask='allowed')

# ============================= PART IV: Check if it is currently nighttime =============================
# Returns whether a waypoints is in nighttime
# Returns 1 if the current-day sunset has passed or the current-day sunrise is yet to come. Else returns 0
def get_is_night(timestamp_prepared, t_sunrise, t_sunset):
    # If the timestamp is earlier than today's sunrise OR after today's sunset
    if (np.datetime64(timestamp_prepared, 'ns') < np.datetime64(t_sunrise, 'ns')) or (
            np.datetime64(timestamp_prepared, 'ns') > np.datetime64(t_sunset, 'ns')):
        is_night = 1
    else:
        is_night = 0

    return is_night

# Applies get_is_night to dataset
def apply_get_is_night(ds_timestamp_segments_prepared, ds_segment_t_sunrise, ds_segment_t_sunset):
    return xr.apply_ufunc(np.vectorize(get_is_night),
                          ds_timestamp_segments_prepared,
                          ds_segment_t_sunrise,
                          ds_segment_t_sunset,
                          dask='allowed')

# ============================= PART V: Check if the segment is a nighttime segment =============================
# Returns whether a waypoint is a nighttime waypoint
# Returns 1 if it is night and the time to sunrise is larger than the threshold time to sunrise. Else returns 0
def get_is_night_segment(nighttime, time_to_sunrise, t_th, contrail_atr20):
    # if the contrail ATR20 has a value smaller than zero
    if contrail_atr20 < 0:
        is_night_segment = 0*nighttime
    # If the time until sunrise is larger than the given threshold time to sunrise
    elif np.timedelta64(time_to_sunrise, 'ns') > np.timedelta64(int(t_th * 3.6e12), 'ns'):
        is_night_segment = 1*nighttime
    else:
        is_night_segment = 0*nighttime

    return is_night_segment

# Applies  get_is_night_segment to dataset
def apply_get_is_night_segment(ds_nighttime, ds_segment_time_to_sunrise, t_th, ds_contrail_atr20):
    return xr.apply_ufunc(np.vectorize(get_is_night_segment),
                          ds_nighttime,
                          ds_segment_time_to_sunrise,
                          t_th,
                          ds_contrail_atr20,
                          dask='allowed')

# ============================= PART VI: Execute the embedded functions =============================
# Part I
ds_timestamp_prepared = apply_prepare_timestamps_for_waypoints(ds_airtraf)
ds_loc_prepared = prepare_loc_for_suntimes(ds_airtraf)
ds_timestamp_segments_prepared = apply_prepare_timestamps_for_segments(ds_airtraf)

# Part II
ds_t_sunrise = apply_get_time_of_sunrise(ds_loc_prepared, ds_timestamp_prepared)
ds_t_sunset = apply_get_time_of_sunset(ds_loc_prepared, ds_timestamp_prepared)
ds_next_day_t_sunrise = apply_get_next_day_time_of_sunrise(ds_loc_prepared,
ds_timestamp_prepared)

# Find the time of sunrise and sunset per segment by averaging the time of its end
waypoints
ds_segment_t_sunrise = ds_t_sunrise.drop_isel(AirTraf_waypoints_out=-1)+(
ds_t_sunrise.drop_isel(AirTraf_waypoints_out=0)-ds_t_sunrise.drop_isel(
AirTraf_waypoints_out=-1))/2
ds_segment_t_sunset = ds_t_sunset.drop_isel(AirTraf_waypoints_out=-1)+(ds_t_sunset.
drop_isel(AirTraf_waypoints_out=0)-ds_t_sunset.drop_isel(AirTraf_waypoints_out=-1))
/2

# Part III
ds_time_to_sunrise = apply_get_time_to_sunrise(ds_timestamp_prepared, ds_t_sunrise,
ds_next_day_t_sunrise)
ds_segment_time_to_sunrise = (ds_time_to_sunrise.drop_isel(AirTraf_waypoints_out=-1)
+ds_time_to_sunrise.drop_isel(AirTraf_waypoints_out=0))/2

# Part IV
ds_nighttime = apply_get_is_night(ds_timestamp_segments_prepared,
ds_segment_t_sunrise, ds_segment_t_sunset)

# Part V
ds_nighttime_segments = apply_get_is_night_segment(ds_nighttime,
ds_segment_time_to_sunrise, t_th, ds_airtraf["routes_out"].isel(AirTraf_properties
=13).drop_isel(AirTraf_waypoints_out=0) * 0.9654676258992805)

return ds_nighttime_segments


def get_average_pareto_front_waypoints_no_extra_cooling(f00_airtraf_folder,
f05_airtraf_folder, f100_airtraf_folder, t_th):
"""
======================================================================================"

" Function that returns the delta specific effective contrail ATR20 and SOC
separately  "
" for day and night without extra cooling. Also a function connecting the dots is
    "
" provided.
    "
" Inputs: 3x file path = 3x *airtraf_ac.nc files
    "
"          t_th
    "

"======================================================================================
"""
# Open the airtraf_ac.nc files
ds_airtraf_00 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f00_airtraf_folder))
ds_airtraf_05 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f05_airtraf_folder))
ds_airtraf_100 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f100_airtraf_folder))

# Correct for 12 h time step by removing duplicates if necessary
if ds_airtraf_00['time'].values[0].astype(str)[11:13] == '12':
    ds_airtraf_00 = ds_airtraf_00.drop_isel(time=np.arange(0, len(ds_airtraf_00["
routes_out"]), 2).tolist())
if ds_airtraf_05['time'].values[0].astype(str)[11:13] == '12':
    ds_airtraf_05 = ds_airtraf_05.drop_isel(time=np.arange(0, len(ds_airtraf_05["
routes_out"]), 2).tolist())
if ds_airtraf_100['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_100 = ds_airtraf_100.drop_isel(time=np.arange(0, len(
ds_airtraf_100["routes_out"]), 2).tolist())

# Get per segment contrail ATR20
per_segment_contrail_atr20_00 = get_per_segment_contrail_atr20(ds_airtraf_00["
routes_out"])
per_segment_contrail_atr20_05 = get_per_segment_contrail_atr20(ds_airtraf_05["
routes_out"])
per_segment_contrail_atr20_100 = get_per_segment_contrail_atr20(ds_airtraf_100["
routes_out"])

# Get the per_segment SOC
per_segment_soc_00 = get_per_segment_soc(ds_airtraf_00["routes_out"])
per_segment_soc_05 = get_per_segment_soc(ds_airtraf_05["routes_out"])
per_segment_soc_100 = get_per_segment_soc(ds_airtraf_100["routes_out"])

# -------------- REPLACE EXTRA COOLING SEGMENTS ---------------- #
# The SOC optimal segment is used, only if the +0.5% SOC segment is cooler and
cooling.
# Otherwise the +0.5% SOC segment is used.
treated_per_segment_contrail_atr20_05 = per_segment_contrail_atr20_00.where((
per_segment_contrail_atr20_05 < per_segment_contrail_atr20_00) & (
per_segment_contrail_atr20_05 < 0), per_segment_contrail_atr20_05)

# The treated +0.5% SOC segment is used, only if the climate optimal segment is
cooler and cooling.
# Otherwise the climate optimal segment is used.
treated_per_segment_contrail_atr20_100 = treated_per_segment_contrail_atr20_05.where
((per_segment_contrail_atr20_100 < treated_per_segment_contrail_atr20_05) & (
per_segment_contrail_atr20_100 < 0), per_segment_contrail_atr20_100)

# Where the contrail atr20 was replaced, also replace SOC
treated_per_segment_soc_05 = per_segment_soc_05.where(per_segment_contrail_atr20_05
== treated_per_segment_contrail_atr20_05, per_segment_soc_00)

treated_per_segment_soc_100 = per_segment_soc_100.where(
per_segment_contrail_atr20_100 == treated_per_segment_contrail_atr20_100,
treated_per_segment_soc_05)

# -------------- DISTINCT NIGHT FROM DAY --------------- #
# Get witch segments are during daytime
nighttime_segments_00 = get_nighttime_segments(ds_airtraf_00, t_th)
nighttime_segments_05 = get_nighttime_segments(ds_airtraf_05, t_th)
nighttime_segments_100 = get_nighttime_segments(ds_airtraf_100, t_th)

# Also treat nighttime/daytime for exclusion of extra cooling
treated_nighttime_segments_05 = nighttime_segments_05.where(
per_segment_contrail_atr20_05 == treated_per_segment_contrail_atr20_05,
nighttime_segments_00)

treated_nighttime_segments_100 = nighttime_segments_100.where(
per_segment_contrail_atr20_100 == treated_per_segment_contrail_atr20_100,
treated_nighttime_segments_05)

# Get witch segments are during daytime
daytime_segments_00 = (nighttime_segments_00 - 1) * -1
treated_daytime_segments_05 = (treated_nighttime_segments_05 - 1) * -1
treated_daytime_segments_100 = (treated_nighttime_segments_100 - 1) * -1

# Get the night contrail ATR20 per segment
night_per_segment_contrail_atr20_00 = per_segment_contrail_atr20_00 *
nighttime_segments_00
treated_night_per_segment_contrail_atr20_05 = treated_per_segment_contrail_atr20_05
* treated_nighttime_segments_05
treated_night_per_segment_contrail_atr20_100 =
treated_per_segment_contrail_atr20_100 * treated_nighttime_segments_100

# Check if any negative nighttime contrail ATR20 occurred
any_negative_night_contrail_atr20_00 = night_per_segment_contrail_atr20_00.where(
night_per_segment_contrail_atr20_00 < 0)
print("00 negative_night_contrail_ATR20_contributions ",
any_negative_night_contrail_atr20_00.count().values)
treated_any_negative_night_contrail_atr20_05 =
treated_night_per_segment_contrail_atr20_05.where(
treated_night_per_segment_contrail_atr20_05 < 0)
print("05 negative_night_contrail_ATR20_contributions ",
treated_any_negative_night_contrail_atr20_05.count().values)
treated_any_negative_night_contrail_atr20_100 =
treated_night_per_segment_contrail_atr20_100.where(
treated_night_per_segment_contrail_atr20_100 < 0)
print("100 negative_night_contrail_ATR20_contributions ",
treated_any_negative_night_contrail_atr20_100.count().values)

# Get the day contrail ATR20 per segment
day_per_segment_contrail_atr20_00 = per_segment_contrail_atr20_00 *
daytime_segments_00
treated_day_per_segment_contrail_atr20_05 = treated_per_segment_contrail_atr20_05 *
treated_daytime_segments_05
treated_day_per_segment_contrail_atr20_100 = treated_per_segment_contrail_atr20_100
* treated_daytime_segments_100

# Get the night SOC per segment
night_per_segment_soc_00 = per_segment_soc_00 * nighttime_segments_00
treated_night_per_segment_soc_05 = treated_per_segment_soc_05 *
treated_nighttime_segments_05
treated_night_per_segment_soc_100 = treated_per_segment_soc_100 *
treated_nighttime_segments_100

# Get the day SOC per segment
day_per_segment_soc_00 = per_segment_soc_00 * daytime_segments_00
treated_day_per_segment_soc_05 = treated_per_segment_soc_05 *
treated_daytime_segments_05
treated_day_per_segment_soc_100 = treated_per_segment_soc_100 *
treated_daytime_segments_100

# --------------- NORMALIZE W.R.T. SEGMENT EFFECTIVE KM --------------- #
effective_distance_00 = get_effective_per_segment_km(ds_airtraf_00["routes_out"])
effective_distance_05 = get_effective_per_segment_km(ds_airtraf_05["routes_out"])
effective_distance_100 = get_effective_per_segment_km(ds_airtraf_100["routes_out"])

# Also treat effective km for exclusion of extra cooling
treated_effective_distance_05 = effective_distance_05.where(
per_segment_contrail_atr20_05 == treated_per_segment_contrail_atr20_05,
effective_distance_00)

treated_effective_distance_100 = effective_distance_100.where(
per_segment_contrail_atr20_100 == treated_per_segment_contrail_atr20_100,
treated_effective_distance_05)

# Get the normalized night contrail ATR20 per segment
n_night_per_segment_contrail_atr20_00 = night_per_segment_contrail_atr20_00 /
effective_distance_00
treated_n_night_per_segment_contrail_atr20_05 =
treated_night_per_segment_contrail_atr20_05 / treated_effective_distance_05
treated_n_night_per_segment_contrail_atr20_100 =
treated_night_per_segment_contrail_atr20_100 / treated_effective_distance_100

# Get the normalized day contrail ATR20 per segment
n_day_per_segment_contrail_atr20_00 = day_per_segment_contrail_atr20_00 /
effective_distance_00
treated_n_day_per_segment_contrail_atr20_05 =
treated_day_per_segment_contrail_atr20_05 / treated_effective_distance_05
treated_n_day_per_segment_contrail_atr20_100 =
treated_day_per_segment_contrail_atr20_100 / treated_effective_distance_100

# Get the normalized night SOC per segment
n_night_per_segment_soc_00 = night_per_segment_soc_00 / effective_distance_00
treated_n_night_per_segment_soc_05 = treated_night_per_segment_soc_05 /
treated_effective_distance_05
treated_n_night_per_segment_soc_100 = treated_night_per_segment_soc_100 /
treated_effective_distance_100

# Get the normalized day SOC per segment
n_day_per_segment_soc_00 = day_per_segment_soc_00 / effective_distance_00
treated_n_day_per_segment_soc_05 = treated_day_per_segment_soc_05 /
```

```
treated_effective_distance_05
treated_n_day_per_segment_soc_100 = treated_day_per_segment_soc_100 / treated_effective_distance_100

# --------------- GET STANDARD DEVIATION --------------- #
prep_std_nighttime_segments_00 = nighttime_segments_00.where(nighttime_segments_00 != 0, np.nan)
treated_prep_std_nighttime_segments_05 = treated_nighttime_segments_05.where(treated_nighttime_segments_05 != 0, np.nan)
treated_prep_std_nighttime_segments_100 = treated_nighttime_segments_100.where(treated_nighttime_segments_100 != 0, np.nan)

prep_std_daytime_segments_00 = daytime_segments_00.where(daytime_segments_00 != 0, np.nan)
treated_prep_std_daytime_segments_05 = treated_daytime_segments_05.where(treated_daytime_segments_05 != 0, np.nan)
treated_prep_std_daytime_segments_100 = treated_daytime_segments_100.where(treated_daytime_segments_100 != 0, np.nan)

prep_std_per_segment_night_contrail_atr20_00 = n_night_per_segment_contrail_atr20_00 * prep_std_nighttime_segments_00
treated_prep_std_per_segment_night_contrail_atr20_05 = treated_n_night_per_segment_contrail_atr20_05 * treated_prep_std_nighttime_segments_05
treated_prep_std_per_segment_night_contrail_atr20_100 = treated_n_night_per_segment_contrail_atr20_100 * treated_prep_std_nighttime_segments_100

prep_std_per_segment_night_soc_00 = n_night_per_segment_soc_00 * prep_std_nighttime_segments_00
treated_prep_std_per_segment_night_soc_05 = treated_n_night_per_segment_soc_05 * treated_prep_std_nighttime_segments_05
treated_prep_std_per_segment_night_soc_100 = treated_n_night_per_segment_soc_100 * treated_prep_std_nighttime_segments_100

prep_std_per_segment_day_contrail_atr20_00 = n_day_per_segment_contrail_atr20_00 * prep_std_daytime_segments_00
treated_prep_std_per_segment_day_contrail_atr20_05 = treated_n_day_per_segment_contrail_atr20_05 * treated_prep_std_daytime_segments_05
treated_prep_std_per_segment_day_contrail_atr20_100 = treated_n_day_per_segment_contrail_atr20_100 * treated_prep_std_daytime_segments_100

prep_std_per_segment_day_soc_00 = n_day_per_segment_soc_00 * prep_std_daytime_segments_00
treated_prep_std_per_segment_day_soc_05 = treated_n_day_per_segment_soc_05 * treated_prep_std_daytime_segments_05
treated_prep_std_per_segment_day_soc_100 = treated_n_day_per_segment_soc_100 * treated_prep_std_daytime_segments_100

prep_std_night_daily_mean_contrail_atr20_00 = prep_std_per_segment_night_contrail_atr20_00.mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
treated_prep_std_night_daily_mean_contrail_atr20_05 = treated_prep_std_per_segment_night_contrail_atr20_05.mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
treated_prep_std_night_daily_mean_contrail_atr20_100 = treated_prep_std_per_segment_night_contrail_atr20_100.mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)

prep_std_night_daily_mean_soc_00 = prep_std_per_segment_night_soc_00.mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
treated_prep_std_night_daily_mean_soc_05 = treated_prep_std_per_segment_night_soc_05.mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
treated_prep_std_night_daily_mean_soc_100 = treated_prep_std_per_segment_night_soc_100.mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)

prep_std_day_daily_mean_contrail_atr20_00 = prep_std_per_segment_day_contrail_atr20_00.mean(dim=["AirTraf_routes_out", "
AirTraf_waypoints_out"], skipna=True)
treated_prep_std_day_daily_mean_contrail_atr20_05 =
treated_prep_std_per_segment_day_contrail_atr20_05.mean(dim=["AirTraf_routes_out", "
AirTraf_waypoints_out"], skipna=True)
treated_prep_std_day_daily_mean_contrail_atr20_100 =
treated_prep_std_per_segment_day_contrail_atr20_100.mean(dim=["AirTraf_routes_out",
"AirTraf_waypoints_out"], skipna=True)

prep_std_day_daily_mean_soc_00 = prep_std_per_segment_day_soc_00.mean(dim=["
AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
treated_prep_std_day_daily_mean_soc_05 = treated_prep_std_per_segment_day_soc_05.
mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)
treated_prep_std_day_daily_mean_soc_100 = treated_prep_std_per_segment_day_soc_100.
mean(dim=["AirTraf_routes_out", "AirTraf_waypoints_out"], skipna=True)

treated_prep_std_night_delta_daily_mean_contrail_atr20_05 =
treated_prep_std_night_daily_mean_contrail_atr20_05 -
prep_std_night_daily_mean_contrail_atr20_00
treated_prep_std_night_delta_daily_mean_contrail_atr20_100 =
treated_prep_std_night_daily_mean_contrail_atr20_100 -
prep_std_night_daily_mean_contrail_atr20_00

treated_prep_std_night_delta_daily_mean_soc_05 =
treated_prep_std_night_daily_mean_soc_05 - prep_std_night_daily_mean_soc_00
treated_prep_std_night_delta_daily_mean_soc_100 =
treated_prep_std_night_daily_mean_soc_100 - prep_std_night_daily_mean_soc_00

treated_prep_std_day_delta_daily_mean_contrail_atr20_05 =
treated_prep_std_day_daily_mean_contrail_atr20_05 -
prep_std_day_daily_mean_contrail_atr20_00
treated_prep_std_day_delta_daily_mean_contrail_atr20_100 =
treated_prep_std_day_daily_mean_contrail_atr20_100 -
prep_std_day_daily_mean_contrail_atr20_00

treated_prep_std_day_delta_daily_mean_soc_05 =
treated_prep_std_day_daily_mean_soc_05 - prep_std_day_daily_mean_soc_00
treated_prep_std_day_delta_daily_mean_soc_100 =
treated_prep_std_day_daily_mean_soc_100 - prep_std_day_daily_mean_soc_00

treated_std_specific_night_delta_contrail_atr20_05 =
treated_prep_std_night_delta_daily_mean_contrail_atr20_05.std()
treated_std_specific_night_delta_contrail_atr20_100 =
treated_prep_std_night_delta_daily_mean_contrail_atr20_100.std()

treated_std_specific_night_delta_soc_05 =
treated_prep_std_night_delta_daily_mean_soc_05.std()
treated_std_specific_night_delta_soc_100 =
treated_prep_std_night_delta_daily_mean_soc_100.std()

treated_std_specific_day_delta_contrail_atr20_05 =
treated_prep_std_day_delta_daily_mean_contrail_atr20_05.std()
treated_std_specific_day_delta_contrail_atr20_100 =
treated_prep_std_day_delta_daily_mean_contrail_atr20_100.std()

treated_std_specific_day_delta_soc_05 = treated_prep_std_day_delta_daily_mean_soc_05
.std()
treated_std_specific_day_delta_soc_100 =
treated_prep_std_day_delta_daily_mean_soc_100.std()

# --------------- SUM TO A TOTAL --------------- #
# Get total nighttime contrail ATR20
total_nighttime_contrail_atr20_00 = n_night_per_segment_contrail_atr20_00.sum(dim=['
AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
treated_total_nighttime_contrail_atr20_05 =
treated_n_night_per_segment_contrail_atr20_05.sum(dim=['AirTraf_waypoints_out', '
AirTraf_routes_out', 'time'])
treated_total_nighttime_contrail_atr20_100 =
treated_n_night_per_segment_contrail_atr20_100.sum(dim=['AirTraf_waypoints_out', '
AirTraf_routes_out', 'time'])
```

```python

    # Get total daytime contrail ATR20
    total_daytime_contrail_atr20_00 = n_day_per_segment_contrail_atr20_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_daytime_contrail_atr20_05 = treated_n_day_per_segment_contrail_atr20_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_daytime_contrail_atr20_100 = treated_n_day_per_segment_contrail_atr20_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get total nighttime SOC
    total_nighttime_soc_00 = n_night_per_segment_soc_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_nighttime_soc_05 = treated_n_night_per_segment_soc_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_nighttime_soc_100 = treated_n_night_per_segment_soc_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get total daytime SOC
    total_daytime_soc_00 = n_day_per_segment_soc_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_daytime_soc_05 = treated_n_day_per_segment_soc_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_daytime_soc_100 = treated_n_day_per_segment_soc_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # --------------- NORMALIZE W.R.T. NUMBER OF SEGMENTS --------------- #
    # Count number of night segments to later calculate the mean
    n_nighttime_segments_00 = nighttime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_nighttime_segments_05 = treated_nighttime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_nighttime_segments_100 = treated_nighttime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    print("00 nighttime_segments  ", n_nighttime_segments_00.values)
    print("05 nighttime_segments  ", treated_n_nighttime_segments_05.values)
    print("100 nighttime_segments ", treated_n_nighttime_segments_100.values)

    # Count number of day segments to later calculate the mean
    n_daytime_segments_00 = daytime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_daytime_segments_05 = treated_daytime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_daytime_segments_100 = treated_daytime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    print("00 daytime_segments  ", n_daytime_segments_00.values)
    print("05 daytime_segments  ", treated_n_daytime_segments_05.values)
    print("100 daytime_segments ", treated_n_daytime_segments_100.values)

    # Get mean effective specific nighttime contrail ATR20
    mean_nighttime_contrail_atr20_00 = total_nighttime_contrail_atr20_00 / n_nighttime_segments_00
    treated_mean_nighttime_contrail_atr20_05 = treated_total_nighttime_contrail_atr20_05 / treated_n_nighttime_segments_05
    treated_mean_nighttime_contrail_atr20_100 = treated_total_nighttime_contrail_atr20_100 / treated_n_nighttime_segments_100

    print("00 mean_nighttime_contrail_ATR20  ", mean_nighttime_contrail_atr20_00.values, " [K/km]")
    print("05 mean_nighttime_contrail_ATR20  ", treated_mean_nighttime_contrail_atr20_05.values, " [K/km]")
    print("100 mean_nighttime_contrail_ATR20 ", treated_mean_nighttime_contrail_atr20_100.values, " [K/km]")

    # Get mean effective specific daytime contrail ATR20
    mean_daytime_contrail_atr20_00 = total_daytime_contrail_atr20_00 /
```

```
n_daytime_segments_00
treated_mean_daytime_contrail_atr20_05 = treated_total_daytime_contrail_atr20_05 / treated_n_daytime_segments_05
treated_mean_daytime_contrail_atr20_100 = treated_total_daytime_contrail_atr20_100 / treated_n_daytime_segments_100

print("00 mean_daytime_contrail_ATR20  ", mean_daytime_contrail_atr20_00.values, " [K/km]")
print("05 mean_daytime_contrail_ATR20  ", treated_mean_daytime_contrail_atr20_05.values, " [K/km]")
print("100 mean_daytime_contrail_ATR20 ", treated_mean_daytime_contrail_atr20_100.values, " [K/km]")

# Get mean effective specific nighttime SOC
mean_nighttime_soc_00 = total_nighttime_soc_00 / n_nighttime_segments_00
treated_mean_nighttime_soc_05 = treated_total_nighttime_soc_05 / treated_n_nighttime_segments_05
treated_mean_nighttime_soc_100 = treated_total_nighttime_soc_100 / treated_n_nighttime_segments_100

print("00 mean_nighttime_SOC  ", mean_nighttime_soc_00.values, " [$/km]")
print("05 mean_nighttime_SOC  ", treated_mean_nighttime_soc_05.values, " [$/km]")
print("100 mean_nighttime_SOC ", treated_mean_nighttime_soc_100.values, " [$/km]")

# Get mean effective specific daytime SOC
mean_daytime_soc_00 = total_daytime_soc_00 / n_daytime_segments_00
treated_mean_daytime_soc_05 = treated_total_daytime_soc_05 / treated_n_daytime_segments_05
treated_mean_daytime_soc_100 = treated_total_daytime_soc_100 / treated_n_daytime_segments_100

print("00 mean_daytime_SOC  ", mean_daytime_soc_00.values, " [$/km]")
print("05 mean_daytime_SOC  ", treated_mean_daytime_soc_05.values, " [$/km]")
print("100 mean_daytime_SOC ", treated_mean_daytime_soc_100.values, " [$/km]")

# -------------- FIND FRACTIONAL CHANGE W.R.T. SOC OPTIMAL  --------------- #
# Find change in mean effective specific contrail ATR20
delta_nighttime_contrail_atr20_00 = mean_nighttime_contrail_atr20_00 - mean_nighttime_contrail_atr20_00
treated_delta_nighttime_contrail_atr20_05 = treated_mean_nighttime_contrail_atr20_05 - mean_nighttime_contrail_atr20_00
treated_delta_nighttime_contrail_atr20_100 = treated_mean_nighttime_contrail_atr20_100 - mean_nighttime_contrail_atr20_00

delta_daytime_contrail_atr20_00 = mean_daytime_contrail_atr20_00 - mean_daytime_contrail_atr20_00
treated_delta_daytime_contrail_atr20_05 = treated_mean_daytime_contrail_atr20_05 - mean_daytime_contrail_atr20_00
treated_delta_daytime_contrail_atr20_100 = treated_mean_daytime_contrail_atr20_100 - mean_daytime_contrail_atr20_00

# Find change in mean effective specific SOC
delta_nighttime_soc_00 = mean_nighttime_soc_00 - mean_nighttime_soc_00
treated_delta_nighttime_soc_05 = treated_mean_nighttime_soc_05 - mean_nighttime_soc_00
treated_delta_nighttime_soc_100 = treated_mean_nighttime_soc_100 - mean_nighttime_soc_00

delta_daytime_soc_00 = mean_daytime_soc_00 - mean_daytime_soc_00
treated_delta_daytime_soc_05 = treated_mean_daytime_soc_05 - mean_daytime_soc_00
treated_delta_daytime_soc_100 = treated_mean_daytime_soc_100 - mean_daytime_soc_00

# Collect in lists
delta_nighttime_contrail_atr20_lst = [delta_nighttime_contrail_atr20_00, treated_delta_nighttime_contrail_atr20_05, treated_delta_nighttime_contrail_atr20_100]
delta_daytime_contrail_atr20_lst = [delta_daytime_contrail_atr20_00, treated_delta_daytime_contrail_atr20_05, treated_delta_daytime_contrail_atr20_100]

```

```
    delta_nighttime_soc_lst = [delta_nighttime_soc_00, treated_delta_nighttime_soc_05, treated_delta_nighttime_soc_100]
    delta_daytime_soc_lst = [delta_daytime_soc_00, treated_delta_daytime_soc_05, treated_delta_daytime_soc_100]

    treated_std_plus_delta_nighttime_contrail_atr20_lst = [delta_nighttime_contrail_atr20_00,

    treated_delta_nighttime_contrail_atr20_05+treated_std_specific_night_delta_contrail_atr20_05,

    treated_delta_nighttime_contrail_atr20_100+treated_std_specific_night_delta_contrail_atr20_100]
    treated_std_plus_delta_daytime_contrail_atr20_lst = [delta_daytime_contrail_atr20_00,

    treated_delta_daytime_contrail_atr20_05+treated_std_specific_day_delta_contrail_atr20_05,

    treated_delta_daytime_contrail_atr20_100+treated_std_specific_day_delta_contrail_atr20_100]

    treated_std_plus_delta_nighttime_soc_lst = [delta_nighttime_soc_00,
                                            treated_delta_nighttime_soc_05+treated_std_specific_night_delta_soc_05,
                                            treated_delta_nighttime_soc_100+treated_std_specific_night_delta_soc_100]
    treated_std_plus_delta_daytime_soc_lst = [delta_daytime_soc_00,
                                          treated_delta_daytime_soc_05+treated_std_specific_day_delta_soc_05,
                                          treated_delta_daytime_soc_100+treated_std_specific_day_delta_soc_100]

    treated_std_min_delta_nighttime_contrail_atr20_lst = [delta_nighttime_contrail_atr20_00,

    treated_delta_nighttime_contrail_atr20_05-treated_std_specific_night_delta_contrail_atr20_05,

    treated_delta_nighttime_contrail_atr20_100-treated_std_specific_night_delta_contrail_atr20_100]
    treated_std_min_delta_daytime_contrail_atr20_lst = [delta_daytime_contrail_atr20_00,

    treated_delta_daytime_contrail_atr20_05-treated_std_specific_day_delta_contrail_atr20_05,

    treated_delta_daytime_contrail_atr20_100-treated_std_specific_day_delta_contrail_atr20_100]

    treated_std_min_delta_nighttime_soc_lst = [delta_nighttime_soc_00,
                                           treated_delta_nighttime_soc_05-treated_std_specific_night_delta_soc_05,
                                           treated_delta_nighttime_soc_100-treated_std_specific_night_delta_soc_100]
    treated_std_min_delta_daytime_soc_lst = [delta_daytime_soc_00,
                                         treated_delta_daytime_soc_05-treated_std_specific_day_delta_soc_05,
                                         treated_delta_daytime_soc_100-treated_std_specific_day_delta_soc_100]

    # Finds straight lines to make plot more readable
    fit_nighttime = np.polyfit(delta_nighttime_contrail_atr20_lst, delta_nighttime_soc_lst, 2)
    fit_daytime = np.polyfit(delta_daytime_contrail_atr20_lst, delta_daytime_soc_lst, 2)

    func_nighttime = np.poly1d(fit_nighttime)
    func_daytime = np.poly1d(fit_daytime)

    treated_std_plus_fit_nighttime = np.polyfit(
    treated_std_plus_delta_nighttime_contrail_atr20_lst,
    treated_std_plus_delta_nighttime_soc_lst, 2)
    treated_std_plus_fit_daytime = np.polyfit(
    treated_std_plus_delta_daytime_contrail_atr20_lst,
    treated_std_plus_delta_daytime_soc_lst, 2)

    treated_std_plus_func_nighttime = np.poly1d(treated_std_plus_fit_nighttime)
    treated_std_plus_func_daytime = np.poly1d(treated_std_plus_fit_daytime)

    treated_std_min_fit_nighttime = np.polyfit(
    treated_std_min_delta_nighttime_contrail_atr20_lst,
    treated_std_min_delta_nighttime_soc_lst, 2)
    treated_std_min_fit_daytime = np.polyfit(
    treated_std_min_delta_daytime_contrail_atr20_lst,
    treated_std_min_delta_daytime_soc_lst, 2)

    treated_std_min_func_nighttime = np.poly1d(treated_std_min_fit_nighttime)
    treated_std_min_func_daytime = np.poly1d(treated_std_min_fit_daytime)

    # Calculate percentage changes for tops
    max_nighttime_relative_change_contrail_atr20 = np.round((
    treated_delta_nighttime_contrail_atr20_100 / mean_nighttime_contrail_atr20_00 * 100)
    .values)
    max_nighttime_relative_change_soc = np.round((treated_delta_nighttime_soc_100 /
    mean_nighttime_soc_00 * 100).values, 1)
    max_daytime_relative_change_contrail_atr20 = np.round((
    treated_delta_daytime_contrail_atr20_100 / mean_daytime_contrail_atr20_00 * 100).
    values)
    max_daytime_relative_change_soc = np.round((treated_delta_daytime_soc_100 /
    mean_daytime_soc_00 * 100).values, 1)

    # Calculate percentage changes for ~0.5% extra SOC
    med_nighttime_relative_change_contrail_atr20 = np.round((
    treated_delta_nighttime_contrail_atr20_05 / mean_nighttime_contrail_atr20_00 * 100).
    values)
    med_nighttime_relative_change_soc = np.round((treated_delta_nighttime_soc_05 /
    mean_nighttime_soc_00 * 100).values, 1)
    med_daytime_relative_change_contrail_atr20 = np.round((
    treated_delta_daytime_contrail_atr20_05 / mean_daytime_contrail_atr20_00 * 100).
    values)
    med_daytime_relative_change_soc = np.round((treated_delta_daytime_soc_05 /
    mean_daytime_soc_00 * 100).values, 1)

    to_be_returned = (delta_daytime_contrail_atr20_lst,
                      delta_nighttime_contrail_atr20_lst,
                      delta_daytime_soc_lst,
                      delta_nighttime_soc_lst,
                      func_daytime,
                      func_nighttime,
                      max_nighttime_relative_change_contrail_atr20,
                      max_nighttime_relative_change_soc,
                      max_daytime_relative_change_contrail_atr20,
                      max_daytime_relative_change_soc,
                      nighttime_segments_00,
                      treated_nighttime_segments_05,
                      treated_nighttime_segments_100,
                      treated_std_plus_delta_nighttime_contrail_atr20_lst,
                      treated_std_plus_delta_daytime_contrail_atr20_lst,
                      treated_std_plus_delta_nighttime_soc_lst,
                      treated_std_plus_delta_daytime_soc_lst,
                      treated_std_min_delta_nighttime_contrail_atr20_lst,
                      treated_std_min_delta_daytime_contrail_atr20_lst,
                      treated_std_min_delta_nighttime_soc_lst,
                      treated_std_min_delta_daytime_soc_lst,
                      treated_std_plus_func_nighttime,
                      treated_std_plus_func_daytime,
                      treated_std_min_func_nighttime,
                      treated_std_min_func_daytime,
                      med_nighttime_relative_change_contrail_atr20,
                        med_nighttime_relative_change_soc,
                        med_daytime_relative_change_contrail_atr20,
                        med_daytime_relative_change_soc)

    return to_be_returned


def get_potcov_average_pareto_front_waypoints_no_extra_cooling(f00_airtraf_folder,
    f05_airtraf_folder, f100_airtraf_folder, nighttime_segments_00,
    treated_nighttime_segments_05, treated_nighttime_segments_100):
    """
    ======================================================================================"
    " Function that returns the delta specific effective potcov and SOC separately  "
    " for day and night without extra cooling. Also a function connecting the dots is
        "
    " provided.
        "
    " Inputs: 3x file path = 3x *airtraf_ac.nc files
        "
    "        t_th
        "
    "======================================================================================
    """
    # Open the airtraf_ac.nc files
    ds_airtraf_00 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f00_airtraf_folder))
    ds_airtraf_05 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f05_airtraf_folder))
    ds_airtraf_100 = xr.open_mfdataset('{}*airtraf_ac.nc'.format(f100_airtraf_folder))

    # Correct for 12 h time step by removing duplicates if necessary
    if ds_airtraf_00['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_00 = ds_airtraf_00.drop_isel(time=np.arange(0, len(ds_airtraf_00["
    routes_out"]), 2).tolist())
    if ds_airtraf_05['time'].values[0].astype(str)[11:13] == '12':
        ds_airtraf_05 = ds_airtraf_05.drop_isel(time=np.arange(0, len(ds_airtraf_05["
    routes_out"]), 2).tolist())
    if ds_airtraf_100['time'].values[0].astype(str)[11:13] == '12':
            ds_airtraf_100 = ds_airtraf_100.drop_isel(time=np.arange(0, len(
    ds_airtraf_100["routes_out"]), 2).tolist())

    # Get per segment contrail ATR20
    per_segment_contrail_atr20_00 = get_per_segment_contrail_atr20(ds_airtraf_00["
    routes_out"])
    per_segment_contrail_atr20_05 = get_per_segment_contrail_atr20(ds_airtraf_05["
    routes_out"])
    per_segment_contrail_atr20_100 = get_per_segment_contrail_atr20(ds_airtraf_100["
    routes_out"])

    # Get per segment potcov
    per_segment_potcov_00 = get_per_segment_potcov(ds_airtraf_00["routes_out"])
    per_segment_potcov_05 = get_per_segment_potcov(ds_airtraf_05["routes_out"])
    per_segment_potcov_100 = get_per_segment_potcov(ds_airtraf_100["routes_out"])

    # --------------- REPLACE EXTRA COOLING SEGMENTS ---------------- #
    # The SOC optimal segment is used, only if the +0.5% SOC segment is cooler and
    cooling.
    # Otherwise the +0.5% SOC segment is used.
    treated_per_segment_contrail_atr20_05 = per_segment_contrail_atr20_00.where((
    per_segment_contrail_atr20_05 < per_segment_contrail_atr20_00) & (
    per_segment_contrail_atr20_05 < 0), per_segment_contrail_atr20_05)

    # The treated +0.5% SOC segment is used, only if the climate optimal segment is
    cooler and cooling.
    # Otherwise the climate optimal segment is used.
    treated_per_segment_contrail_atr20_100 = treated_per_segment_contrail_atr20_05.where
    ((per_segment_contrail_atr20_100 < treated_per_segment_contrail_atr20_05) & (
    per_segment_contrail_atr20_100 < 0), per_segment_contrail_atr20_100)

    # Where the potcov was replaced, also replace SOC
    treated_per_segment_potcov_05 = per_segment_potcov_05.where(
    per_segment_contrail_atr20_05 == treated_per_segment_contrail_atr20_05,
    per_segment_potcov_00)

    treated_per_segment_potcov_100 = per_segment_potcov_100.where(
    per_segment_contrail_atr20_100 == treated_per_segment_contrail_atr20_100,
    treated_per_segment_potcov_05)

    # --------------- DISTINCT NIGHT FROM DAY --------------- #
    # Get witch segments are during daytime
    daytime_segments_00 = (nighttime_segments_00 - 1) * -1
    treated_daytime_segments_05 = (treated_nighttime_segments_05 - 1) * -1
    treated_daytime_segments_100 = (treated_nighttime_segments_100 - 1) * -1

    # Get the night potcov per segment
    night_per_segment_potcov_00 = per_segment_potcov_00 * nighttime_segments_00
    treated_night_per_segment_potcov_05 = treated_per_segment_potcov_05 *
    treated_nighttime_segments_05
    treated_night_per_segment_potcov_100 = treated_per_segment_potcov_100 *
    treated_nighttime_segments_100

    # Get the day potcov per segment
    day_per_segment_potcov_00 = per_segment_potcov_00 * daytime_segments_00
    treated_day_per_segment_potcov_05 = treated_per_segment_potcov_05 *
    treated_daytime_segments_05
    treated_day_per_segment_potcov_100 = treated_per_segment_potcov_100 *
    treated_daytime_segments_100

    # --------------- NORMALIZE W.R.T. SEGMENT EFFECTIVE KM --------------- #
    effective_distance_00 = get_effective_per_segment_km(ds_airtraf_00["routes_out"])
    effective_distance_05 = get_effective_per_segment_km(ds_airtraf_05["routes_out"])
    effective_distance_100 = get_effective_per_segment_km(ds_airtraf_100["routes_out"])

    # Also treat effective km for exclusion of extra cooling
    treated_effective_distance_05 = effective_distance_05.where(
    per_segment_contrail_atr20_05 == treated_per_segment_contrail_atr20_05,
    effective_distance_00)

    treated_effective_distance_100 = effective_distance_100.where(
    per_segment_contrail_atr20_100 == treated_per_segment_contrail_atr20_100,
    treated_effective_distance_05)

    # Get the normalized night potcov per segment
    n_night_per_segment_potcov_00 = night_per_segment_potcov_00 / effective_distance_00
    treated_n_night_per_segment_potcov_05 = treated_night_per_segment_potcov_05 /
    treated_effective_distance_05
    treated_n_night_per_segment_potcov_100 = treated_night_per_segment_potcov_100 /
    treated_effective_distance_100

    # Get the normalized day potcov per segment
    n_day_per_segment_potcov_00 = day_per_segment_potcov_00 / effective_distance_00
    treated_n_day_per_segment_potcov_05 = treated_day_per_segment_potcov_05 /
    treated_effective_distance_05
    treated_n_day_per_segment_potcov_100 = treated_day_per_segment_potcov_100 /
    treated_effective_distance_100

    # --------------- SUM TO A TOTAL --------------- #
    # Get total nighttime potcov
    total_nighttime_potcov_00 = n_night_per_segment_potcov_00.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_nighttime_potcov_05 = treated_n_night_per_segment_potcov_05.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_nighttime_potcov_100 = treated_n_night_per_segment_potcov_100.sum(dim
    =['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get total daytime potcov
    total_daytime_potcov_00 = n_day_per_segment_potcov_00.sum(dim=['
    AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
```

```python
    treated_total_daytime_potcov_05 = treated_n_day_per_segment_potcov_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_total_daytime_potcov_100 = treated_n_day_per_segment_potcov_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # --------------- NORMALIZE W.R.T. NUMBER OF SEGMENTS --------------- #
    # Count number of night segments to later calculate the mean
    n_nighttime_segments_00 = nighttime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_nighttime_segments_05 = treated_nighttime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_nighttime_segments_100 = treated_nighttime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Count number of day segments to later calculate the mean
    n_daytime_segments_00 = daytime_segments_00.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_daytime_segments_05 = treated_daytime_segments_05.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])
    treated_n_daytime_segments_100 = treated_daytime_segments_100.sum(dim=['AirTraf_waypoints_out', 'AirTraf_routes_out', 'time'])

    # Get mean effective specific nighttime potcov
    mean_nighttime_potcov_00 = total_nighttime_potcov_00 / n_nighttime_segments_00
    treated_mean_nighttime_potcov_05 = treated_total_nighttime_potcov_05 / treated_n_nighttime_segments_05
    treated_mean_nighttime_potcov_100 = treated_total_nighttime_potcov_100 / treated_n_nighttime_segments_100

    print("00 mean_nighttime_potcov  ", mean_nighttime_potcov_00.values, " [km/km]")
    print("05 mean_nighttime_potcov  ", treated_mean_nighttime_potcov_05.values, " [km/km]")
    print("100 mean_nighttime_potcov ", treated_mean_nighttime_potcov_100.values, " [km/km]")

    # Get mean effective specific daytime potcov
    mean_daytime_potcov_00 = total_daytime_potcov_00 / n_daytime_segments_00
    treated_mean_daytime_potcov_05 = treated_total_daytime_potcov_05 / treated_n_daytime_segments_05
    treated_mean_daytime_potcov_100 = treated_total_daytime_potcov_100 / treated_n_daytime_segments_100

    print("00 mean_daytime_potcov ", mean_daytime_potcov_00.values, " [km/km]")
    print("05 mean_daytime_potcov ", treated_mean_daytime_potcov_05.values, " [km/km]")
    print("100 mean_daytime_potcov ", treated_mean_daytime_potcov_100.values, " [km/km]")

    return
```