SDP for Torque Tracking in Real-Time

Author:
Luca Hartmann

Supervisors:
Orcun Karaca
Tinus Dorfling

April 7, 2023

Contents

Chapter 1

Numerical Case Studies

1.1 Performance metrics

The goal of this work is to show the applicability of FCS-MPC for torque tracking in a real world setting. Performance metrics give us an understanding about the tracking and time performance that the proposed algorithms have. We investigate FCS-MPC for torque and absolute stator flux tracking. The tracking performance can be quantified by the root-mean squared (rms) error. That is

$$e_T(k) = ||T^*(k) - T(k)||_2^2$$

for the torque $T(k)$ and

$$e_\Psi(k) = ||\Psi^*(k) - \Psi(k)||_2^2$$

for the absolute stator flux $\Psi(k)$, where $T^*(k)$ and $\Psi^*(k)$ are the torque and absolute stator flux reference, respectively. In addition we measure the cost accumulated by the controller. It is defined as

$$J_{acc}(k) = \sum_{l=0}^{k} J(k).$$

The accumulated cost is a simple way of comparing different controller performances. It will later be used to show how different controllers perform in certain operational points, such as torque steps.

Besides the tracking performance of the controllers, the computational time plays a major role. It can vary strongly depending on the operational conditions of the controller and is a major challenge in non-convex torque tracking. Yet, measuring the solving time in a simulation can be a challenge; unlike an embedded device, a computer can be affected by a variety of working conditions that influence the solving time. To better estimate the computational time on an embedded device, the number of function calls of the recursive branch-and-bound algorithm is taken into consideration. A function call can be interpreted as a parent node in the search tree of the branch-and-bound algorithm. Therefore, in the remainder, we will refer to the number of parent nodes n_p visited to resemble the computational time. For the simulation discussed in Section 1.4.1, we observed a correlation of $\text{Corr}(t, n_p) = 0.9989$ between the computational time t and the number of parent nodes n_p.

1.2 Parameter Choices and Simulation Setting

There are view hyper parameters that have to be chosen for the controller. The cost function contains the torque weighting factor λ_T, which discounts the torque ripple to prioritize the flux magnitude ripple, without changing the cost ratio between these two terms and the switching penalty. We use the same parameter choice $\lambda_T = 0.052$. In addition we have to decide about the switching penalty λ_u. It is chosen to $\lambda_u = 3.5 \times 10^{-3}$ to achieve a switching frequency of approximately $f_{\mathrm{sw}} = 215\,\mathrm{Hz}$. The sampling intervals of all controllers are chosen to $T_{\mathrm{s}} = 100\,\mu\mathrm{s}$ and the horizons are $N = 5$. Additionally, in the case of a parent node limit we use $n_{p,max} = 250$, unless stated otherwise. Table 1.1 contains a list of all controller parameters.

parameter name	parameter symbol	parameter values
torque weighting factor	λ_T	0.052
switching weight	λ_{u}	3.5×10^{-3}
sampling interval	T_{s}	$100\,\mu\mathrm{s}$
horizon	N	5
upper bound on parent nodes	$n_{p,max}$	250
simulation sampling interval	T_{sim}	$0.5\,\mu\mathrm{s}$

Table 1.1: Controller and simulation parameters.

Our comparisons are based on a simulation of a scenario with a ramp-up, step-down, and step-up of the torque. The absolute stator flux reference will be kept at a constant level $\Psi^* = 1$. Figure 1.1 shows the torque reference.

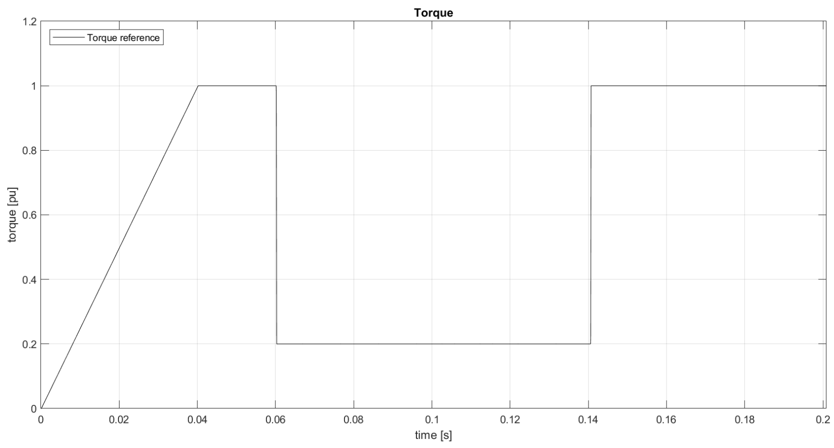

Figure 1.1: Torque reference.

The simulation and FCS-MPC run at different speeds. Throughout all scenarios a simulation sampling interval of $T_{\mathrm{sim}} = 0.5\,\mu\mathrm{s}$ is used. The rotor has a fundamental frequency of $f_1 = 1494\,\mathrm{rpm} = 24.9\,\mathrm{Hz}$. The rotor and stator flux measurements are perturbed by $\eta \sim \mathcal{U}(-2.5 \times 10^{-3},\ 2.5 \times 10^{-3})$, where $\mathcal{U}(a,\ b)$ is the uniform distribution in the interval $[a,\ b]$. All machine parameters are stated in Table 1.2.

parameter name	parameter value
Voltage	3300 V
Current	356 A
Real power	1.587 MW
Apparent power	2.035 MV A
Stator frequency	50 Hz
Rotational speed	1494 rpm
Stator resistance	$R_s = 0.0108$ pu
Rotor resistance	$R_r = 0.0091$ pu
Stator leak. react.	$X_{ls} = 0.1493$ pu
Rotor leak. react.	$X_{lr} = 0.1104$ pu
Main reactance	$X_m = 2.349$ pu
Total leak. react.	$X_\sigma = 0.2548$ pu
dc-link voltage	$V_{dc} = 1.930$ pu
dc-link capacitance	$X_c 11.769$ pu

Table 1.2: Machine parameters.

1.3 Advantages of Multi-Step control

The problem of long solving times of the branch-and-bound algorithm is strongly coupled with the length of the control horizon N. To show that a longer horizon leads to a better performance, we ran simulations with $N \in \{1, 3, 5\}$ and checked its influence on the rms tracking errors of the torque and absolute stator flux. All values are listed in Table 1.3.

Horizon N	$\bar{e}_T$	$\bar{e}_\Psi$	f_{sw}
1	130.3×10^{-3}	9.112×10^{-3}	216.1 Hz
3	80.6×10^{-3}	10.030×10^{-3}	218.2 Hz
5	79.9×10^{-3}	8.865×10^{-3}	215.3 Hz

Table 1.3: Controller average tracking errors for different horizons N and a fixed switching frequency f_{sw}.

It becomes clear that increasing the horizon leads to improved tracking. Comparing $N = 1$ and $N = 3$ shows that there can also be trade offs between torque and absolute stator flux tracking. Nevertheless, choosing $N = 5$ leads to superior tracking for both quantities. Choosing a longer horizon can lead to even better tracking. We will stick to $N = 5$, as is shows clear improvements and still leads to reasonable simulation times.

1.4 Computational feasibility

1.4.1 Drawbacks of Conventional Branch-and-Bound Algorithms

Being non-convex, torque tracking imposes a major challenge on the FCS-MPC formulation. The non-convexity does not allow us to use the sphere decoder discussed in [1]. When deciding upon an input, the sphere decoder has an intuition about the possible future cost. As a result, it is very fast in pruning sub-trees that do not belong to the optimal solution. In [2] the weakness of conventional branch-and-bound algorithms can be seen. We can achieve a major decrease in computational time, when a more meaning-full lower bound is introduced. It helps the decoder to estimate the minimum cost in the future, when applying a certain input. In the discussed case a speed-up of more than three times on average and up to twenty times in single cases was achieved.

In [2] the branch-and-bound algorithm was applied to a non-linear function for upper-bounding the converter's switching frequency. The major advantage was the existence of a simple lower bound on the switching frequency; if there is no switching applied. In the case of torque tracking such a lower bound does not exist. The applicability for estimating future branch costs via SDP relaxations for torque tracking will be further discussed in Section 1.4.2.

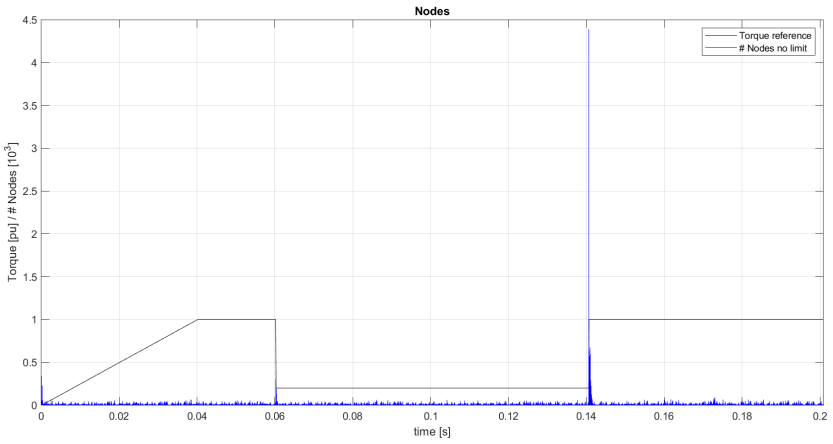

Figure 1.2: Number of traversed parent nodes n_p.

Figure 1.2 depicts the number of traversed nodes for a simulation with a conventional branch-and-bound algorithm. The educated guess is used as an initial solution and there is no limit on the number of parent nodes that are traversed. The torque reference is shown to give an intuition about the current operational condition of the controller. We can observe a high peak during the torque step-up. In this case the number of traversed parent nodes reaches up to $n_p = 4297$. When excluding all time steps with $n_p \geq 250$ we can calculate an average of $\bar{n}_p \approx 19.4$ with a standard deviation of $\text{std}(n_p) \approx 18.2$. This simulation shows the central problem of the convectional branch-and-bound algorithm: The number of parent nodes n_p tends to explode in some operational conditions.

In [3] and [1] different methods for initializing the branch-and-bound algorithm were discussed. We will compare three different initial solutions to obtain an intuition about the relevance of a good initialization.

1. *Bad guess* is a vector of zeros. It represents a bad initial solution.

2. *Ed guess* is the educated guess.

3. *Opt guess* is the optimal solution computed by the previous algorithms. It serves as a benchmark for how good an initial solution can at most be.

Table 1.4 shows the mean, standard deviation, and maximum value of a separate steady-state simulation. Ed guess and opt guess always stay below 250 parent nodes with a simulated maximum of 100 and 65 parent nodes, respectively. The bad guess continuously traverses the highest number of parent nodes. Its maximum number of nodes is $n_{p,bad\ guess} = 497$ parent nodes. This shows that having a good initial guess is important during steady-state to keep

Initial solution	mean	std	max
Bad guess	208.2	88.1	497
Ed guess	18.5	12.9	100
Opt guess	16.1	9.6	65

Table 1.4: Simulated number of nodes traversed for different initial guesses. *mean*, *std*, and *max* refer to the mean, standard deviation, and maximum of all measurements in steady-state. All measurements are in number of parent nodes n_p.

the solving time small. The ed guess is a good enough estimate, as the number of nodes is still far below our maximum bound of 250 nodes.

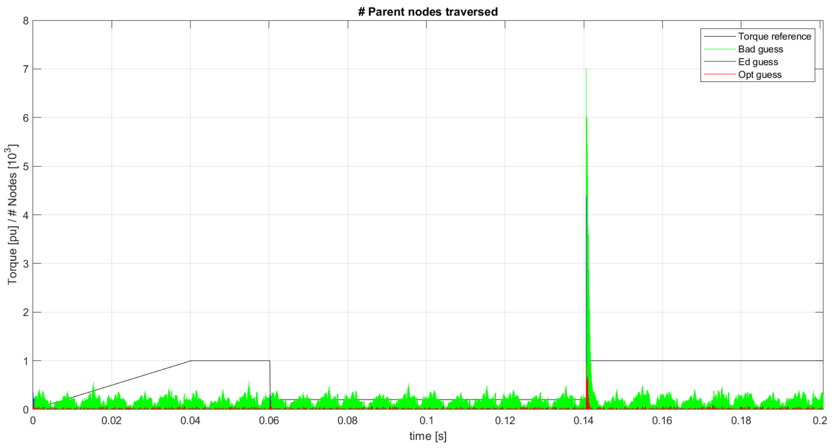

Figure 1.3: Number of traversed parent nodes n_p. Comparison of branch-and-bound algorithms without node limits, but different initial guesses.

In contrast to steady-state operation, we observe a high peak in the number of nodes during the torque step-up for all initial solutions. Figure 1.3 shows the number of parent nodes traversed for each initial guess. The measured number of visited parent nodes increase up to $n_{p,bad\ guess} = 6948$, $n_{p,ed\ guess} = 4318$, and $n_{p,opt\ guess} = 914$, respectively. All of these values are too high to be computed in real time on an embedded device. The reason for these high numbers during step-up compared to ramps and step-downs is the value of the cost function. The *No node limit* benchmark in Figure 1.4 shows that the torque takes longer to follow the reference during step-up compared to step-down or continuous reference tracking. The bad tracking during the step-up leads to much higher values of the cost function. Therefore, the missing intuition of the branch-and-bound algorithm about future nodes makes it traverse many nodes on non-optimal paths before pruning the respective sub-trees.

Figure 1.4 shows the torque tracking of the two controllers. Both are initialized with an educated guess and execute two branch-and-bound algorithms. The first controller is executed without an upper node limit to give an optimal benchmark. The second controller is executed with a node limit of 250 parent nodes. We can observe that both controllers behave very similarly until the torque step-up. During the step-up the controller with an upper node limit performs much worse than the benchmark. This shows that cutting the branch-and-bound algorithm due to time constraints can lead to suboptimality and deteriorate the controller's performance.

We can conclude that branch-and-bound algorithms are well suited for FCS-MPC with torque tracking during most operational modes, such as ramps, steady-state, and step-downs.

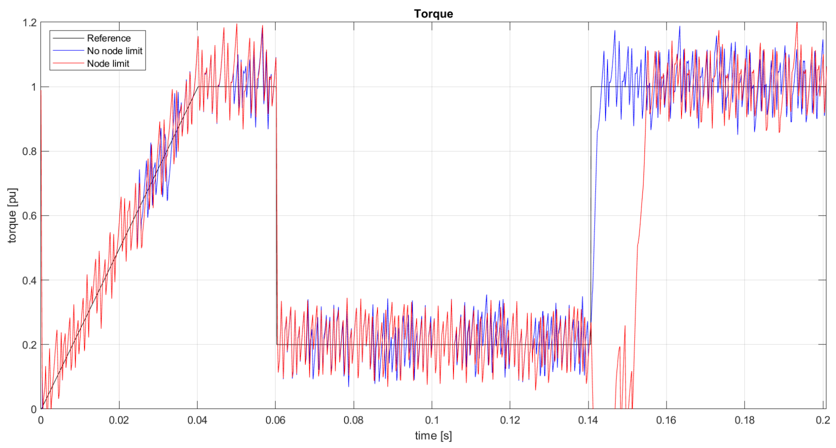

Figure 1.4: Torque of systems controlled by controllers with an educated guess as initial solution. *No node limit* has no node limit and *Node limit* has a node limit of 250 nodes.

Some operational conditions, such as torque step-ups lead to a great number of parent nodes that need to be traversed to find an optimal solution. They make branch-and-bound algorithms badly suited for solving the FCS-MPC's optimization problem. This raises the question on how to solve for good-enough solutions in real time during these operational conditions.

1.4.2 Applicability of different speed-ups

We focus on solving one SDP relaxation per controller sample to speed-up solving during reference step-ups. On average, directly applying the SDP guess does not lead to a good average performance compared to the educated guess with a branch-and-bound algorithm (see Table 1.1). Nevertheless, the SDP relaxation has a constant solving time in every operational mode, when fixing the number of gradient descent iterations. Therefore, it is able to compute a good-enough solution, when the branch-and-bound algorithm fails due to its varying solving time. We thought of the following two algorithms that apply an SDP guess:

1. *Ed & sdp guess* is a branch-and-bound algorithm with an upper limit of 250 parent nodes. But it starts with the best of the educated guess and the SDP guess.

2. *Ed guess + sdp* is a branch-and-bound algorithm that uses an educated guess, with an upper limit of 250 parent nodes. Additionally, an SDP guess is computed in parallel. In the final step, the best of the two is chosen.

To benchmark improvements, we introduce the conventional FSC-MPC; *Ed guess* is a branch-and-bound algorithm that starts with an educated guess, with an upper limit of 250 parent nodes. Both, Ed & sdp guess and Ed guess + sdp can potentially be executed in real time. Solving the SDP relaxation takes longer than obtaining the educated guess. It is difficult to estimate the number of parent nodes that corresponds to the additional computational complexity of the SDP guess. This makes a comparison of Ed guess and Ed & sdp guess difficult. Thus, we use Ed guess + sdp to benchmark improvements compared to Ed guess . Computing the SDP guess in parallel does not influence the number of traversed parent nodes in branch-and-bound and therefore, eases the comparison.

Note that Ed & sdp guess has an at least as good solution as Ed guess + sdp , as the branch-and-bound algorithm can only improve its initial solution. Thus, Ed & sdp guess gives a lower bound for the performance of both algorithms.

Note, we could also improve the branch-and-bound algorithm by giving an estimate of the future cost in a branch. This would closer relate to the speed-up of [2], discussed in Section

1.4.1. SDP relaxations can be used to get a lower bound on the future branch cost. This method was so far not implemented, due to the higher computational effort of solving an SDP relaxation multiple times in a controller sample. During simulations we have observed that getting a realistic guess takes at least 250 gradient descent iterations, when using the first-order scs solver [4]. Lets assume that an SDP would be executed only in the root-note. In the worst case it will be used 27 times, which would lead to 6750 gradient descent iterations in addition to the normal branch-and-bound execution. So far, we do not expect a good-enough trade off between the resulting reduction of visited parent nodes and executed gradient descent iterations to solve the branch-and-bound algorithm in real time.

1.4.3 Performance of algorithms

We will now apply the above-mentioned Ed guess + sdp to see the benefit for solve an SDP relaxation once a controller sample. A comparison between Ed guess + sdp , Ed guess and *Opt* is done, where *Opt* is a branch-and-bound algorithm that starts with an educated guess. It does not have an upper node limit and serves as an optimal benchmark.

Figure 1.4 showed a comparison of Opt and Ed guess , which where referred to as *No node limit* and *Node limit*, respectively. Figure 1.5 additionally shows the torque that is observed for Ed guess + sdp . All controllers behave very similarly until the step-up. At the step-up Ed guess performs badly, while Ed guess + sdp follows the reference about as good as the benchmark Opt . A similar behavior can be observed in the absolute stator flux tracking, depicted in Figure 1.6. This behavior can also be seen in the accumulated cost depicted in Figure 1.7. It shows how the cost of Ed guess accumulates significantly more than the other algorithms during the step-up. Figure 1.8 shows the number of nodes that are visited by the different algorithms. We can observe the high number of parent nodes that Opt traverses, while Ed guess and Ed guess + sdp are cut at 250 parent nodes. Ed guess reaches the limit $n_p = 250$ for longer time than Ed guess + sdp . The reason is that the SDP helps Ed guess + sdp to follow the reference fast during step-up, while Ed guess gets stuck for a while. This shows that the SDP approximation is able to achieve a close-to-optimal performance when the branch-and-bound algorithm is not able to.

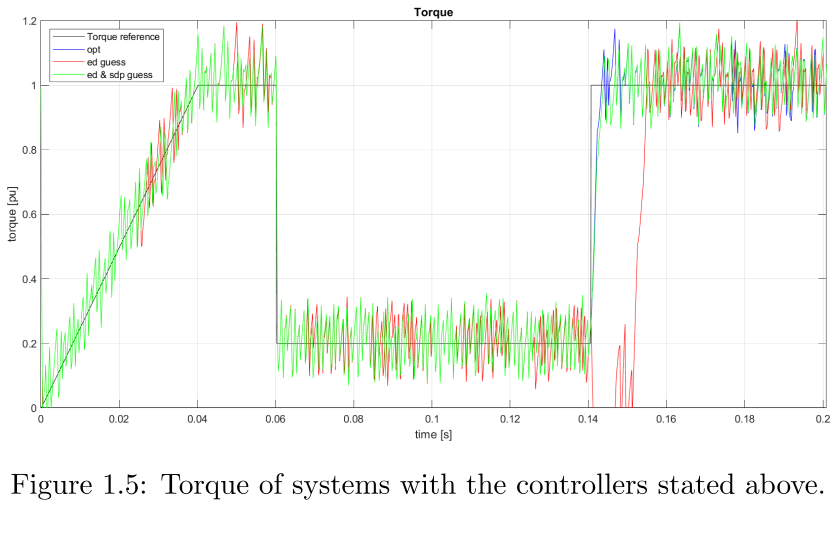

Figure 1.5: Torque of systems with the controllers stated above.

Figures 1.5 to 1.8 were all computed for 250 gradient descent iterations to solve the SDP relaxation. To compare this performance to a possible optimum, we ran the same simulation with an upper limit of 500 gradient descent iterations. The resulting torques and accumulated costs Ed & sdp guess and Ed guess + sdp approach the dynamics of Opt even closer in this

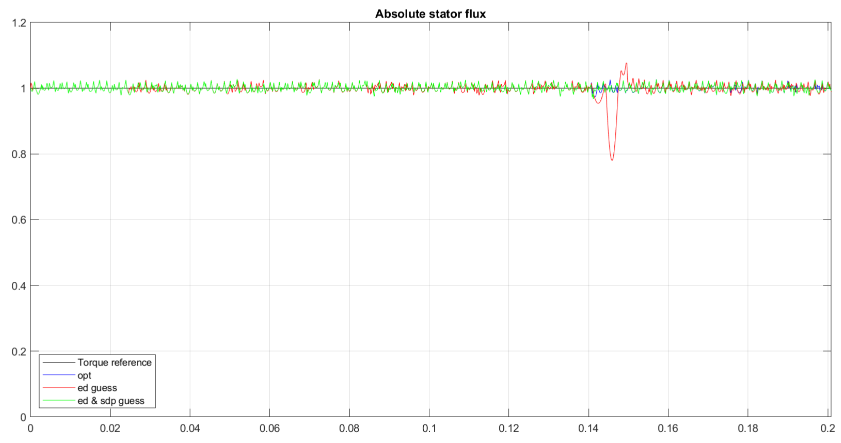

Figure 1.6: Absolute stator flux of systems with the controllers stated above.

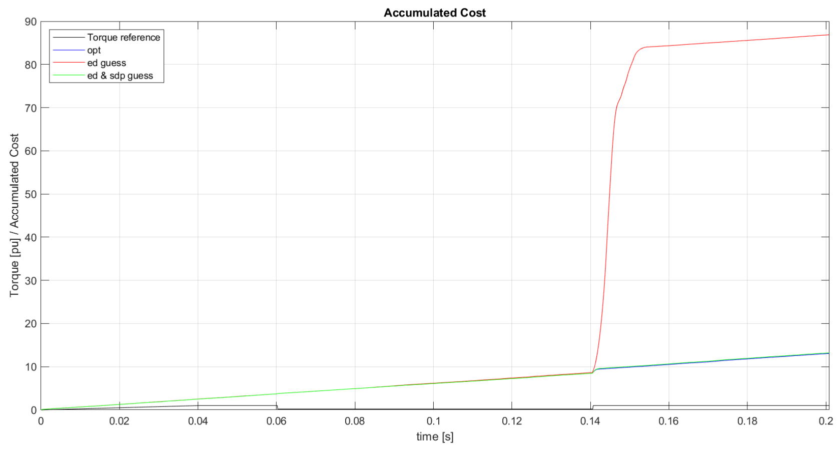

Figure 1.7: Accumulated costs of the controllers stated above.

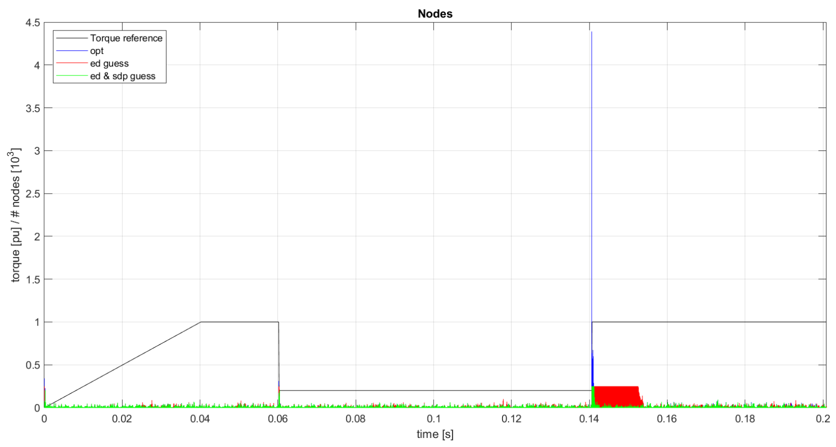

Figure 1.8: Number of traversed parent nodes n_p. Comparison of branch-and-bound algorithms with different node limits and different initial guesses.

case. This leads to the conclusion that already applying as many iterations in the SDP solver leads to close-to-optimal performance.

Bibliography

[1] M. Dorfling, T. Mouton, and T. Geyer, “Selective harmonic suppression for finite-control-set model predictive control,” *21st European Conference on Power Electronics and Applications (EPE ’19 ECCE Europe)*, pp. P.1—-P.10, 2019.

[2] “Non-linear sphere deconding algorithm for switching frequency tracking (/semester project report),”

[3] T. Geyer, *Model Predictive Control of High Power Converters and Industrial Drives.* 09 ed., 2016.

[4] B. O’Donoghue, E. Chu, N. Parikh, and S. Boyd, “Conic optimization via operator splitting and homogeneous self-dual embedding,” *Journal of Optimization Theory and Applications*, vol. 169, pp. 1042–1068, June 2016.