A dynamic programming approach for optimizing train speed profiles with speed restrictions and passage points

Haahr, Jørgen Thorlund; Pisinger, David; Sabbaghian, Mohammad

Published in:
Transportation Research Part B: Methodological

Link to article, DOI:
10.1016/j.trb.2016.12.016

Publication date:
2017

Document Version
Peer reviewed version

Link back to DTU Orbit

Citation (APA):
Haahr, J. T., Pisinger, D., & Sabbaghian, M. (2017). A dynamic programming approach for optimizing train speed profiles with speed restrictions and passage points. *Transportation Research Part B: Methodological*, *99*, 167-182. https://doi.org/10.1016/j.trb.2016.12.016

A Dynamic Programming Approach for Optimizing Train Speed Profiles with Speed Restrictions and Passage Points

Jørgen Thorlund Haahr
Technical University of Denmark
jhaa@dtu.dk

David Pisinger
Technical University of Denmark
dapi@dtu.dk

Mohammad Sabbaghian
Cubris ApS
m.sabbaghian@cubris.dk

Abstract

This paper considers a novel solution method for generating improved train speed profiles with reduced energy consumption. The solution method makes use of a time-space graph formulation which can be solved through Dynamic Programming. Instead of using uniform discretization of time and space as seen previously in the literature, we rely on an event-based decomposition that drastically reduces the search space. This approach is very flexible, making it easy to handle, e.g., speed limits, changes in altitude, and passage points that need to be crossed within a given time window. Based on solving an extensive number of real-life problem instances, our benchmarks show that the proposed solution method is able to satisfy all secondary constraints and still be able to decrease energy consumption by 3.3% on average compared to a commercial solver provided by our industrial collaborator, Cubris. The computational times are generally very low, making it possible to recompute the train speed profile in case of unexpected changes in speed restrictions or timings. This is a great advantage over static offline lookup tables. Also, the framework is very flexible, making it possible to handle a number of additional constraints on robustness, passenger comfort etc. Selected details of the method and benchmark are only described at a high level for confidentiality reasons.

1 Introduction

Railway operation provides a fast and efficient mode of transportation. Although being very energy efficient compared to most other transportation forms, the overall energy consumption is large. Three of the largest railway network operators in Europe together spend up to €1.75 billion annually on energy [2].

In Germany, for example, railway traffic consumes 2% of the country's energy usage. Thus, operators have strong incentives to reduce energy usage while maintaining punctuality. Passenger train services are often operated daily in tight preplanned schedules while a number of long and heavy freight trains are planned in a more *ad hoc* basis. Even minor energy reductions in individual train schedules can lead to significant savings. Studies have shown that potential savings of $5 - 20\%$ are possible by optimizing train speed profiles [16].

Regardless of whether it is a freight or passenger train service, the train travels from a source station to a destination station, possibly making intermediate stops. Timeslots on tracks are allocated to the train allowing passage through the rail infrastructure, and strict timing is imposed on arrivals and departures to enforce customer satisfaction and a high network utilization. A train driver thus knows in advance when he has to depart from the current station and arrive on the next. Furthermore, *passage points* in between stipulate individual time-windows when the train must pass certain areas.

Given a maximum trip time $T_{\max}$ the speed profile, i.e., how fast the train is driving throughout the trip, is mainly determined by the train driver. This is a complex task, so in order to avoid delays the driver often decides to drive as fast as possible, using unnecessarily much energy. The fastest speed profile may also in some cases be infeasible due to passage point restrictions, e.g. due to the interlocking system for track-changes or other conflicts. Examples of different speed profiles are illustrated in Figure 1. Essentially, a speed profile consists of up to four different actions: acceleration, braking, cruising and coasting. Acceleration and braking can be performed with 0–100% effect, but research has shown that only maximum acceleration and maximum braking can be used in an optimal journey. Although braking can, in some cases, regenerate energy we will not consider this aspect. Utilizing the regenerated energy is another non-trivial problem.

Energy usage corresponds to the fuel or electricity consumption of the train and is often measured by the traction effort of the train over some trip. A train driver directly controls the consumption by regulating the speed throttle. In order to save energy the train driver can operate the train at lower speeds, thus loosing less energy to natural resistances, or perform coasting. There is a clear trade-off between trip duration and energy consumption, in general, faster traversal requires more energy per kilometer. Analytical formulas can project the effect of increasing or reducing speed, however combining a complete profile is not trivial. Multiple additional constraints further complicate the problem, e.g., changing speed restrictions, passage point time requirements and changes in altitude. We will refer to this optimization problem as the Train Speed Profile Problem (TSPP).

In this paper we propose a novel Dynamic Programming (DP) methodology for optimizing the TSPP in real-time that is suitable for an on-board Driver Advisory System (DAS). The method is in a sense elegant and does not rely on advanced software packages or mathematical solvers. In addition to gradient forces and speed-restrictions we show how the method can handle passage points. Passage points are frequently occurring constraints that requires the train to

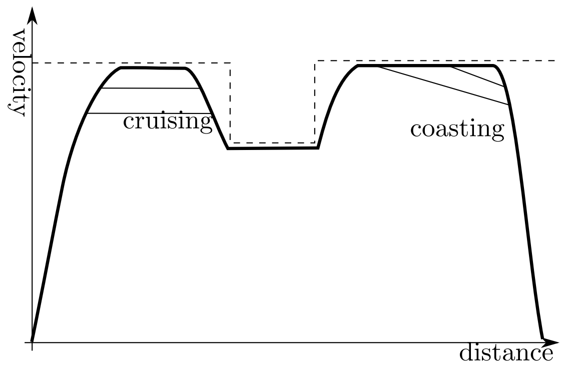

Figure 1: The plot shows simple examples of feasible speed profiles. Speed-limits are marked with dashed lines. The bold path illustrates an aggressive strategy following the speed-limits as closely as possible by using full acceleration and braking, thus arriving as soon as possible. Alternative and more energy efficient profiles can be achieved, at the expense of a longer trip time, by lower cruise speed or by performing coasting as shown. The available travel time, $T_{\max}$, limits the number of feasible profiles.

pass certain points (e.g. signals or intersections) in specific time windows. This type of constraints were proposed by our industrial collaboration partners as their problem instances include such restrictions. For instance, the important *train separation problem* [6], ensuring that two trains have sufficient distance between them to avoid collision, can be modeled by use of passage points. Two trains traveling along the same track are commonly separated by signals. Each train has to pass the signal within a specified time windows. The passage point may be crossed with any legal velocity, making the constraint harder to handle than full stops.

The solution method is based on a space-velocity graph representation of partial speed profiles that are computed using equations for train motion. Compared to previous dynamic programming approaches we do not uniformly discretize space nor time. An advantage of the underlying graph representation is the ability to extend the number of possible choices without changing the solver. The approach is not guaranteed to be optimal, unless all necessary partial speed profiles are generated, but, intelligently selecting the partial profiles, this method is able to find very high quality solutions. The choices represented in the underlying graph are optimized using a label setting algorithm that minimizes energy usage subject to a time resource constraint. Note, that given an initial (possible high quality) solution, the construction of the graph can be improved by considering similar but alternative driving choices. Thus, if the method is used in an iterative loop, where the output solution is the input of the next iteration, then the method can be used to converge towards a locally optimal solution.

We benchmark our solution approach on multiple real-life instances from different train operators. The instances include multiple actual speed-restrictions, passage points and altitude profiles. We show that the computational times are sufficiently small for producing continuous real-time speed-advice. The results are compared to a fast solution method, denoted *C-solver*, used by several train operators and kindly provided by our industrial partner. For the purpose of this study, we compare both approaches in a static environment between two train stops.

In this work some important assumptions are made. Firstly, we assume that the train is able to hold speed (maintaining a constant velocity), e.g., using cruise control. Feasible solutions (speed profiles) will therefore contain segments with a constant velocity. Even if this is not an option, we assume that a post-processing step can be performed that approximates a constant speed by switching between acceleration and coasting. We note that the trains used in our experimental study are able to hold speed. Secondly, we assume that there exist no steep track slopes. Without such the train has sufficient engine power to accelerate or hold speed at any point. In addition the train has sufficient braking capabilities to decelerate or hold speed at any point. In effect, the train can never be stuck, e.g., if operating at too low or high velocities respectively. We note however, that the given framework can, without affecting the time complexity of the algorithm, be extended to include steep hills. Our case studies contain no such steep track slopes; such is the case in many established railway infrastructures. Thirdly, we assume that only a discrete set of cruising speeds can be used, since it is impractical to give speed advice of non-rounded numbers to a train driver. This is a practical constraint given by our industry partner. However, with a larger resolution the method can approximate a continuous set of cruising speeds. The assumption also limits the solution space as only a discrete set of cruising speeds can be selected.

1.1 Related Literature

Much research is conducted in the area of determining optimized speed profiles. Theoretical results were established already a few decades ago [8, 17, 31], and methods to optimize speed profiles are proposed by many authors in different settings. In the following we briefly mention some fundamental theoretical work and the related work on numerical solution methods. We refer to Hansen et al. [16] for an introduction to train running time estimation and energy-efficient train operation. Albrecht et al. [3, 4] present a two-part summary and review in optimal train control theory. A recent review on energy-efficient train operation, with emphasis on urban rail transit, is given by Yang et al. [36]. They present an overview of energy optimization of the timetabling, speed profile generation and the integration of both problems. Another review of both analytical and numerical approaches for determining optimal speed profiles is presented by Want et al. [33]. Finally, [32] present a method for train speed optimization based on Mixed Integer Programming. The model approximates the nonlinear motion functions by piece-wise affine functions. The reported soluton times are large, in the magnitude of 10 minutes.

Heuristic methods for generating speed profile optimization are proposed in literature with different extensions. However, in contrast to our contribution, none of these are based on the same underlying structure for a Dynamic Programming algorithm that is used in this paper. Our approach is flexible as it is to a large extent independent of the underlying equations for train motion and cost structure. Furthermore, the algorithm is able to handle passage points. Some track segments are shared with multiple trains which requires synchronization of train passage. Each train must therefore transfer from one block to the next within a specified time-window. Passage points can also be used to increase robustness. To the best of our knowledge the only paper dealing with passage points is the paper by Albrecht et al [6] but only in the context of train separation. However, there is currently no explicit and efficient numerical computaiton scheme to implement the results in [6].

Fundamental theoretical work is established by Howlett and Pudney [20]. In [18] they show that an optimal driving strategy must have certain properties. With continuous speed control they show, using the Pontryagin principle, that the optimal speed profile consists of a power-hold-coast-brake strategy. The simplified optimization problem thus consists of identifying the optimal switching points. However, with multiple speed restrictions and steep hills this strategy must contain multiple power-hold-coast-brake sequences. Cheng et al. [10] study analytical approaches for handling multiple speed limits with continuous speed.

The theoretical work is extended by Howlett et al. [19] to include steep sections, i.e., track segments where the train is unable to maintain or accelerate velocity due to a relatively high climb, and vice versa for downhill slopes. They show how to find the optimal switching points. Passage points are not considered, but they claim that it is possible to extend the work to include speed limits. In a later work, Albrecht et al. [5] prove that there is only one optimal strategy when considering switching points for steep sections. Several solution methods in literature are based on finding optimal switching points to construct speed profiles [11, 23, 27].

Successful analytic methods based on real-time computational algorihtms are described in [3, 4, 5, 19, 23, 27]. These methods derive accurate speed profiles by numerically solving solutions of equations of motion and calculate optimal switcing points. A commercial implementation of these methods, *Energymiser* is, among otheres, being used in Australasia, England and France.

However, apart from [6], no theoretical work considers passage points, and how this affects the optimal driving strategy. Further, no solution methods are found in literature that include passage point restrictions. In contrast to the departure and arrival point, the velocity at passage points is not fixed. The problem entails determining when to arrive at the passage point (within a given time-window) and at which velocity.

Franke et al. [14] propose a DP approach for solving the TSPP. They model multiple speed restrictions, regenerative braking and gravitational forces on slopes. In addition, they consider a more realistic modelling of engine efficiency and power losses in the propulsion system. They do, however, not model passage points. Implementational and benchmarking details of the DP algorithm are omitted but they show that the approach provides significant gains over very simple strategies in the one studied test case.

A short study of another DP programming approach is presented by Ko et al. [24] for solving a variation of the TSPP. They include complicating constraints that provide a more realistic model of the electric motive force and regenerative braking. Time is discretized uniformly in the approach and they show one benchmark with multiple speed restriction and track slopes. A solution is found within 22 seconds for the study and they conclude that the approach is acceptable for practical purposes.

In addition to an Ant Colony Optimization (ACO) and a Genetic Algorithm (GA) approach Lu et al. [28] also consider a DP approach. The solution space is heuristically restricted in order to reduce the large state-space and thereby improving both computational time and memory usage. A lattice grid is adopted to reduce the feasible region, and choices that deviate too much from an average speed are pruned. The methods are benchmarked against a single instance using three different journey times. Multiple speed restrictions and altitude data are included. The DP approach obtains the best results, but the computational time requirement is close to one hour.

Another DP approach is proposed and tested by Larranaga et al. [25]. They consider extensions to the problem using more realistic curves of the tractive and brake efforts. Short journeys of a metro system are used as a case study. In the DP method, time, space and velocity are discretized uniformly and the solution method is able to solve a short trip within one hour of computational time. All states are computed and recorded which means that the results can be used as a lookup table which can be used if the actual speed profile deviates from the optimized one. They conclude that a fine discretization is needed in order to obtain accurate results. In contrast, we include passage points as additional constraints and aim for computational times usable in a real-time system. The DP presented in our work does not discretize space nor time uniformly and states are only evaluated if they can lead to an optimal solution.

Several heuristic methods for solving the TSPP have been presented in literature. We briefly mention some of the most important studies here. Jong et al. [22] present a heuristic based on a logic flow-diagram used to identify promising solutions. They compare it to a commercial software package using realistic train journeys provided by the Taiwan Railway Administration. Computational times and solution quality are competitive. Wong et al. [35] consider the TSPP without the possibility to cruise at constant speed. They present a few methods for solving the problem by determining either one or multiple coasting points, i.e., with or without speed restrictions. One short and one long trip instance are considered where the solution methods require respectively around 90 and 350 seconds to solve. Chang and Sim [9] present a GA for producing a driving profile lookup table for an automated train operation controller. Multiple speed restrictions are included and a more detailed model for regenerative braking is used. No computational times are reported for the one considered instance. A GA approach has been considered in many instances in literature [15, 34, 35].

Albrecht [7] argue that train speed optimization should be handled as an integrated part of train dispatching. Li et al. [26] present an integrated solution approach for the timetabling and speed profile problem. Departure times are considered together with the selected speed profiles for the cyclic timetable of a single metro train line service in Beijing. The experiments reveal a significant improvement over existing results when considering the net energy consumption. The solution approach is heuristic and based on a Genetic Algorithm methodology. The train dynamics are simplified as the approach assumes no running resistances, a constant acceleration and constant deceleration. Real-life data from the train line services are benchmarked. No CPU run-times are reported.

Finally, Pacciarelli and Pronzo [29] consider the problem of determining feasible speed profiles for a number of trains circulating in a given area. The algorithm makes use of a feed-back system between the speed regulator system and the conflict resolution system.

1.2 Overview

The structure of this paper is as follows. In Section 2, we present a problem definition for the train speed profile problem and outline important assumptions. Then, we present a graph representation of the problem in Section 3 which is the input to our solver. We show how to construct the problem graph using partial profiles of acceleration, braking, cruising and coasting. The dynamic programming solver is then presented in Section 4, where we discuss and present an outline of the algorithm used to solve the graph problem instances. Finally, results are presented in Section 5 comparing our algorithm with a state-of-the-art commercial train advice system. Section 6 presents some better bounds on the solution. Section 7 shows how various additional constraints easily can be incorporated in the framework, and our conclusions are summarized in Section 8.

2 Problem Definition

Assume that the considered train, according to a given timetable, has to depart from one station and arrive at the next station at a specific time. The initial and final speed is equal to zero. The train is assumed to depart at time 0, and a maximum total trip time, $T_{\max} \geq 0$, is given. The distance to travel is determined by the track-layout between the stations and consist of changing track slopes. Although not necessary, assume that the altitude profile consist of piece-wise linear slopes. Different speed restrictions may exist on the trip, i.e., the whole trip is partitionable into a consecutive sequence of segments with changing speed limits. Assume that no lower bound is imposed on the speed limit, although it is easy to extend the model to handle this case. If the velocity of the train exceeds the upper limit, at any point in time, then that profile is infeasible. A train must brake in advance before reaching a new segment if the speed limit there is lower than the current velocity.

Assume that the train has continuous speed throttle control, and the ability to hold a certain velocity once it has been reached. Trains with discrete controls are a different line of study, but we note that a hold-speed (i.e. cruising speed) phase can be approximated in a post-processing step. Assume, in this work, that no steep hill exists on the trip that makes it impossible to hold a certain velocity. Note, however, that the solution method we propose can, without affecting the time complexity of the overall algorithm, be extended to include such. The characteristics of the train are known, such as the total mass, acceleration power, service braking capability and resistance parameters. Although not a limiting constraint, assume that no regenerative braking can occur. Regenerating power while braking the train has been technically possible for some time, but how to efficiently use that power is non-trivial when optimizing one train speed profile in isolation.

Multiple passage points can exists on the trip. A passage point refers to a specific location (between the arrival and departure station) that must be passed in a given time window. A speed profile that passes this location earlier or later than the time window is considered infeasible. No additional speed restrictions, other than the governing ones, are enforced. It is for example not required for a train to come to a full stop at the location. Passage points can be considered constraints that e.g. mimic signal lights where the train can only pass while the signal is green. Passage points introduce additional complexity to the traditional TSPP. A solution method is now also forced to consider the distribution of runtime surplus between consecutive passage points.

Any speed profile that respects the constraints mentioned above is feasible. The optimal speed profile is defined to be the feasible profile that consumes the least energy. Other interesting solution characteristics, not considered in the paper, could be included such as a *punctual* arrival or the overall robustness of the profile. Passage time windows, especially the final destination, state that any speed profile that arrives within a certain threshold of the planned time is feasible; however, it may be more attractive to arrive more punctually rather than saving a small amount of energy. Figure 2 shows an example of the TSPP.

We define the fastest speed profile to be the profile that requires minimal time usage (in our case also ignoring passage point restrictions). This profile is unique and can be generated by accelerating the train as fast as possible to the active speed limit (or the maximum train speed), only braking just in time for lower speed limits.

A solution method for the defined problem can be used as a responsive on-board DAS, provided that solutions can be found very quickly. Such real-time systems are extremely useful for coping with variations in the daily schedules (e.g. cargo/passenger weight or climate) as well as larger disturbances (e.g. changed passage timings and trip times). For the sake of simplicity of the presentation, we present the problem of optimizing from a full stop to the next stop, but with slight modification this assumption can be lifted in the solution method. The solver can start from any node in space-speed graph. The solver can likewise end in any vertex by removing any vertex from consideration that cannot reach it. Hence it easy to find an optimal speed profile from any starting og ending place.

There are many arguments in favor of a real time on-board algorithm, as

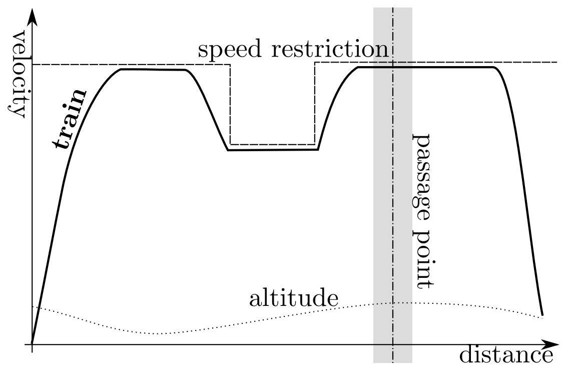

Figure 2: An example of a problem instance. The figure shows a distance and velocity plot for a feasible train profile. Governing speed limits are illustrated as dashed lines, and the altitude profile is shown with a dotted line. The passage points requires the train to pass through a certain point within a certain time window. The bold black path depicts the fastest speed profile.

opposed to pre-calculated speed profiles. The most important reason is the inherent variation in the process. There are many reasons why the actual journey is different from the planned journey; examples are temporary speed restrictions, emergency speed restrictions, route or material changes, and train defects (reduced power). Especially, a Traffic Management System, which solves disruptions by retiming and rerouting trains in real time, will make the speed profile prone to last minute changes. The reason to calculate the advice on-board is to minimize the length of the control-loop[12]. This is the time from detecting the deviation (from the time path) and the actual change of the driver action, i.e., receive a GPS signal, send to central, compare time, recalculate, send back, update screen, driver reaction time and etc. If this loop is too long, the advice has already become obsolete and recalculation is necessary. Calculating the advice on-board and within a second allows the driver the most possible time to adjust the train speed according to the calculated speed profile.

3 Graph Representation

The proposed solution method is essentially a general framework that consists of three main ingredients, or modules: formulas to model the train dynamics, a graph representation, and a solver. One of the main benefits of our approach is the loose coupling between the adopted model for train dynamics and the method for solving the resulting graph problem. In the following we explain the individual components. We adopt state of the art modeling for the train dynamics, and a DP method as solution framework.

3.1 The Train Dynamics

Modelling train dynamics such as acceleration curves and resistances is necessary in order to project a speed profile. The accuracy depends on the chosen assumptions and simplifications. A balance between the achievable computational time and accuracy is needed. In the following we describe the formulas used in this paper. The equations for modelling train motion are adopted from the related literature. The equations and solutions have further been verified by our collaboration partner using their own deployed software solution.

As mentioned earlier, the optimal speed profile for a single trip consists of a power-hold-coast-brake strategy [20]. The acceleration (i.e. power) and braking phases utilize full acceleration capacity and maximal allowed braking. Intuitively, doing either at partial capacity only results in loss of trip buffer time. Assume that the motion of the train can be reduced to the motion of a point mass [20]. Long trains with distributed mass are reducible to a point mass by modifying the altitude profile accordingly.

In the coasting phase no energy is applied to accelerate the train, thus at a certain speed only vehicle resistances such as friction, mechanical and air affect the train speed. We adopt a quadratic formula, that has been used for decades, for modeling such resistances:

$$R(v) = A + B \cdot v + C \cdot v^2$$

The coefficients A, B and C are train specific. Better known as the Davis Equation[13], it states that the resistance is speed-dependent modeled by a quadratic function. In the coasting phase, this equation can be used to describe how much kinetic energy is lost over time or distance to resistances. Given a certain starting or ending speed, it is possible to derive how fast the velocity changes, or when the train reaches a certain target velocity. In addition to the gravitational pull, this equation models the coasting phase.

Gravitational pull is determined by the track slope:

$$F_{grav} = m \cdot g \cdot \sin(\alpha)$$

Where m is the train mass, $g \approx 9.8 {}^{m}/{}_{s^2}$ is the gravitational acceleration and α is the angle of inclination.

Thus the following determines the net force during coasting:

$$F_{coast} = R(v) + F_{grav}$$

During the hold phase, speed remains constant. Energy must be applied by the engine in order to overcome train resistance and gravitational pull. In case of a downward slope, it may not be necessary to apply any energy at all, and potentially requires initiating partial braking.

In the braking phase, we assume that the brakes can apply a constant force. Due to reasons such as safety, equipment tear-and-wear and passenger comfort a fraction of the full braking capacity is adopted, i.e., service braking force, F_{sb}.

$$F_{brake} = F_{sb} + R(v) + F_{grav}$$

Acceleration over time or distance is approximated using the following equation:

$$F_{acc} = \min\left(\frac{F_{engine}}{v}, F_{amax}\right) + R(v) + F_{grav}$$

Where F_{engine} is the maximum engine tractive force. A fairly accurate description of the engine acceleration force is F_{engine}/v, but at low speeds it has proven to be inaccurate [20] - due to physical factors such as adhesion and overheating. F_{amax} denotes the maximal acceleration that is possible for the train at low speeds. Acceleration is naturally also affected by train resistances and gravity.

In this paper, we assume that energy consumption (in terms of fuel or electricity) is proportional to the mechanical energy consumed. Thus, calculating the energy usage is a matter of accumulating the work required by all acceleration and cruising phases. If we considered regenerative braking, the deceleration force would reproduce some of that energy. Although not difficult to include, this aspect is not pursued in this work.

The forces define the motion of train as function of the position [3, Section 1.2]. In the following section, the formulas above will be used to generate partial profiles, and it is central to find intersections between these partial profile. This can be done analytically using the above formulas or estimated numerically. We use a numeric approach as this can be fast using various search methods (e.g. Newtons method) and it adds flexibility to the framework since the partial profiles only need to be differentiable.

We assume the altitude profile is piecewise linear, i.e., the whole trip is partitioned into segments, each having a constant track slope. A trip is thus split into segments defined by *split points* $s_0, s_1 \ldots s_n$. This assumption simplifies the above formulas and provides reasonable accuracy. Accuracy is adjustable by adding more split points.

3.2 Speed Profile Graph

The formulas presented in Section 3.1 can trace partial acceleration, braking, cruising or coasting profiles in a velocity by distance plot. We now present a graph that is generated based on such partial paths where the intersection between these partial profiles define the vertices of the graph. An infinite number of partial paths exist, but we select a subset of these, which are most likely to appear in an optimized speed profile.

The main principle of the graph representation is for every split point s_i and speed u_{ij}, to generate forward and backward speed profiles, corresponding to the four modes(acceleration, braking, cruising, coasting), as briefly outlined in Figure 3.

Not all forward and backward speed profiles are generated, but only those which have a large probability for being chosen according to some heuristic rules: From the initial position project acceleration profiles and whenever the speed limit is increased do the same. Project braking profiles from the final destination and from decreasing speed restrictions. From braking curves project coasting profiles backwards. Finally, cruising profiles are inserted at the predetermined

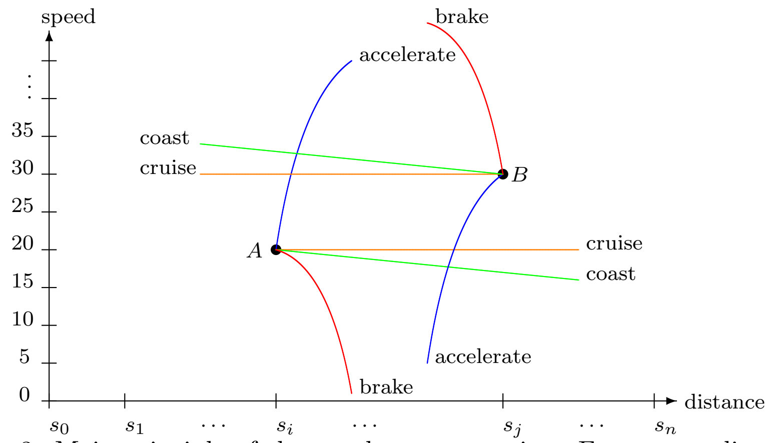

Figure 3: Main principle of the graph representation. For every split point s_i and speed u_{ij}, forward and backward speed profiles are generated corresponding to the four modes. Where forward and backward speed profiles intersect, new nodes are inserted. Forward profiles are illustrated for node $A = (s_i, 20)$ while backward profiles are illustrated for node $B = (s_j, 30)$. Where the forward and backward speed profiles intersect, new nodes are inserted. All relevant accelerate, cruise and brake profiles are generated, while coasting profiles are generated heuristically. Only coasting profiles that will hit the braking curve at one of the discrete speed steps (0 km/h, 5 km/h, 10 km/h ...) are considered.

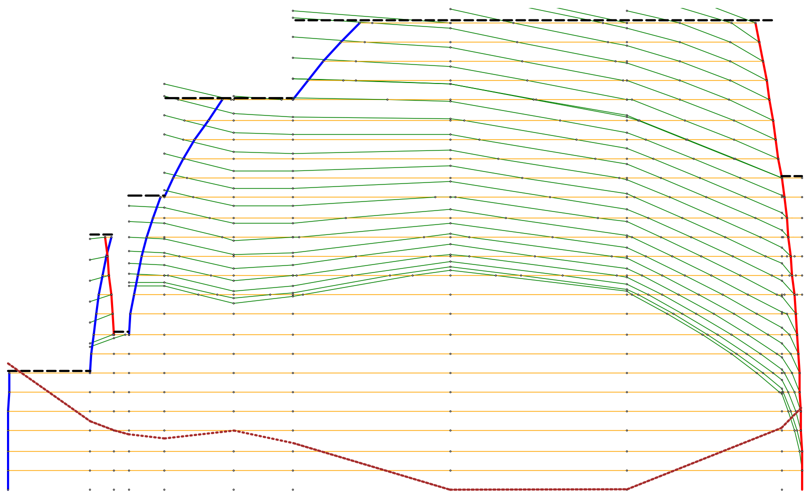

Figure 4: An example of a generated speed profile graph. Speed limits are depicted using solid black dashed lines, and the altitude profile using brown dotted lines. The other lines, which are edges in the graph, depict the partial acceleration(blue), braking (red), cruising (yellow), and coasting (green) profiles.

discretized set of velocities. Figure 4 shows an example of the resulting graph. The altitude profile is shown as a dotted line; observe the gravitational pull on the coasting trajectories. Note, for the sake of simplicity, we add vertices at the border of each region. Vertices with only one edges, or only one *in* and *out* edge can be removed in a preprocessing step. Performing further preprocessing reduces the graph, leading to an average reduction of roughly 70%.

A central concept in the graph is the *region*. A trip is first partitioned into a list of consecutive regions, where each region defines a stretch with identical characteristics. This simplifies some of the analytical formulas, but also identifies key positions. The regions are directly related to the split points defined earlier as two split points define the borders of each individual region. In addition to tracks slopes, we extend the set of split points to include the start and end point of speed restrictions. Note, that the regions are not partitioned uniformly by distance (or time) in general. A change in track slope or maximum speed restriction defines a new region.

As mentioned, one important goal is to give drivers rounded speed advice, e.g. 5, 10, 15 or 20 kilometers per hour. Thus, only a predetermined discrete set of velocities will be considered as feasible cruising speeds. The driver will therefore only receive advice to accelerate or brake to one of these selected rounded velocities. Following fractional speed advice is more impractical to realize by the drivers, thus rounded velocities are, in this sense, more robust. Furthermore, rounding exact speed profiles in a post-procedure is also non-optimal as this will cause the train to arrive at the wrong time if the driver follows the speed advice exactly as instructed. Such a profile is unstable as drivers will face difficulties following it.Discretized speeds have also been applied in early versions of *Energymiser* technology which used precomputed speed profiles for a discrete set of hold speeds [3, 4, 5, 19, 23, 27].

With multiple speed limits, it is not trivial to identify the optimal solution nor the partial paths that it consists of. It is however known that the solution has certain properties. Acceleration should be performed as quickly and early as possible in order to save time. Similarly, braking should be performed as quickly and late as possible in order to save time. Intuitively and informally, late acceleration or early braking does not save energy, and thus only results in an increased trip time without energy gains. Acceleration and braking profiles are therefore inserted at region borders whenever the speed limit changes.

Multiple cruising stages may exist, and identifying the optimal velocity for each cruising stage is part of the problem. As described, a selected discrete set of velocities are allowed, thus we include these as partial profiles. Starting from velocity 0 one partial cruising profile is inserted for every x velocity, where x is the adopted step, determined by the operator. We note that the speed limit is respected while generating these profiles.

A rule set for coasting is more difficult to determine, which is why we rely on a heuristic approach for choosing when to start coasting. However, from the optimal strategy we only need to consider coasting before a braking stage. For every braking curve we therefore adopt a number of partial coasting profiles that intersect with the braking curve.

As a heuristic choice we decided to generate one partial coasting profile that intersects with the braking curve for each of the selected discrete speeds, i.e., at the end of a coasting phase the driver will be told to brake when he hits one of the preselected rounded speeds.

We can consider the constructed graph $G(V, A)$ as a directed graph. Intersections in the graph define the vertices, V, of the graph. The segments of partial paths between two vertices define the arcs, A, of the graph. By construction the graph is acyclic as arcs only connect vertices strictly forward in time. Associated to each vertex $v \in V$ is a position and a speed, and to each arc $a \in A$ is a time and energy consumption. An arc represents a driving choice for a period of time, e.g., accelerating from one position and velocity to another. We define the source, $v_s \in V$, as the initial position at zero velocity, and the sink, $v_t \in V$, as the final destination at zero velocity. We let $outEdges(v) \subseteq A$ denote all arcs originating from $v \in V$.

Any path from v_s to v_t represents a driving profile but the path is not necessarily feasible. Although the actions represent a feasible sequence of choices they will not necessarily respect the time requirements. If the accumulative time of the arcs on the path is less than the total time available, then we say it is a feasible path.

3.2.1 Passage Points

Passage points impose a time interval in which the train should pass the given point. They can among others be used to model signals, e.g., needed to ensure correct separation between trains [6]. Passage points are a natural extension to the proposed graph structure. Firstly, positions of the passage points are now also splitting points in the graph, resulting in an increase in the total number of regions. At these points the train is required to pass in the specified time window. Note, that the final destination is essentially also a passage point. An example graph including passage points is shown in Figure 5. Compared to the previous example in Figure 4 additional acceleration curves can be observed.

In this extension, additional acceleration curves are added such that all of the cruising speeds are reachable before passing the passage point. The additional acceleration curves are needed in the special case where a large excess of time is available before the passage point. In order to save energy a low speed should be selected, however depending on the next region, it might not be feasible to cross the point with a low velocity.

The passage point extension naturally imposes additional constraints to a feasible path. In addition to finding a connected path, all passage points must be reached within specific time windows.

4 Label Setting Algorithm

We use the train speed profile graph in Section 3.2 as a basis for finding optimized profiles. Any path from the source to the target vertex represents a feasible sequence of actions, however it is not necessarily feasible as the available time windows may be violated. Therefore, the TSPP problem has to be solved as a Resource Constrained Shortest Path Problem (RCSPP). This is a well-studied problem and we refer to Irnich et al. [21] for a more in-depth description of how to solve this problem.

We propose a label setting algorithm to solve the RCSPP, which essentially is a DP approach. Label setting algorithms have been applied in literature to efficiently solve multiple variants of the RCSPP. The considered problem structure has an optimal substructure property, but instead of computing a table of all states we are only interested in the optimal solution and therefore can discard inferior states. In contrast to previous DP approaches, we do not generate a lookup table for the whole trip but rely on re-computation when the train deviates from the original plan. We limit this section to a brief description of the label setting algorithm. Note, that using this methodology we find an optimal path in the generated RCSPP, but the solution is only sub-optimal for the TSPP since not all possible transition states are present in the graph.

In the RCSPP, the goal is to minimize the overall energy consumption while respecting the available trip time, $T_{\max}$. The available time can be considered a resource, which is consumed when traversing the arcs in the graph. A label, l, is a sequence of connected arcs in the graph, i.e., a partial path. The energy consumption, $cost(l)$, and time usage, $time(l)$, correspond to the accumulative energy and time usage of the entailed arcs. The source vertex, $v_s \in V$, is the head of any label l and we let $tail(l)$ denote the tail. Note, that for any arc $a \in A$, $cost(a)$, $time(a)$ and $tail(a)$ are defined similarly.

The algorithm is essentially a full enumeration of all possible paths, starting from v_s. The first label simply consists of the source node, having zero cost and no time usage. In a recursive manner, new labels are generated for every possible extension of current label. In the end, all possible partial paths originating at v_s will be generated. However, enumerating all possible paths is far from practical for our problem instances. In order to solve the instances efficiently search branches are pruned and discarded in the process if they prove to be fruitless. We adopt the use of infeasible labels and dominance to efficiently prune the states in the enumeration process.

Theorem 1 (Infeasible Label I)**.** *Infeasible labels can be pruned. A label l is infeasible if $time(l) > T_{\max}$.*

Theorem 2 (Infeasible Label II)**.** *Infeasible labels by time bound can be pruned. A label l is infeasible by time bound if $time(l)+\delta > T_{\max}$ where δ is the minimum time required to reach v_t.*

Theorem 3 (Dominated Label)**.** *A dominated label can be pruned. Consider two labels a and b at the same vertex (i.e. having same speed and location). Label l_a dominates label l_b if $cost(l_a) \leq cost(l_b)$ and $time(l_a) = time(l_b)$.*

Notice, that it may not be an advantage to arrive earlier to a vertex since one may arrive too early to a passage point. If no passage points are present one may use the stronger dominance rule saying that label l_a dominates label l_b if $cost(l_a) \leq cost(l_b)$ and $time(l_a) \leq time(l_b)$.

Algorithm 1

```
1: procedure LabelSettingAlgorithm($G(V, A), T_{\max}$)    ▷ graph and trip time
2:     $\mathbf{L} \leftarrow \{$CreateLabel$(v_s, 0, 0, \emptyset)\}$    ▷ stack of unprocessed labels
3:     $\mathbf{B} \leftarrow \emptyset$    ▷ best solution
4:     while $|\mathbf{L}| \geq 0$ do
5:         $l \leftarrow$ Front($\mathbf{L}$)
6:         $\mathbf{L} \leftarrow$ PopFront($\mathbf{L}$)    ▷ Remove $l$ from unprocessed list
7:         $v \leftarrow tail(l)$
8:         ApplyDominance($v$)    ▷ pruning dominated vertices
9:         for $a \in outEdges(v)$ do    ▷ performing extensions
10:            $v' \leftarrow tail(a)$
11:            $l' \leftarrow$ CreateLabel($v, cost(l) + cost(a), time(l) + time(b), l$)
12:            if IsFeasible($l'$) then
13:                $\mathbf{L} \leftarrow$ PushBack($\mathbf{L}, l'$)
14:                if $v' = v_t \ \wedge \ cost(l') < cost(\mathbf{B})$ then
15:                    $\mathbf{B} \leftarrow l'$
16:                end if
17:            end if
18:        end for
19:    end while
20:    return B
21: end procedure
```

The general label setting algorithm is summarized in Algorithm 1. Starting with the source vertex, labels are processed in iteration by making all possible extensions. Infeasible extensions are pruned from consideration, and dominance is checked in order to prune fruitless labels.

A list of unprocessed labels is maintained in the algorithm. All labels in the list are unprocessed, i.e., their extensions have to be considered. Initially, only one label is in the list. This label is the source vertices (v_s), i.e., the start of any path. In every iteration the frontmost label is extracted (**PopFront**) from the list. One new label is generated for every possible extensions (i.e. arc) from it's current vertex. The new labels are put on the back of the list. The current vertex of the new labels corresponds to the destination vertex of the arc, energy and time are accumulated. Without dominance and feasibility check the list grows exponentially. If a label is infeasible, there is no reason to continue down that path, hence it is not considered any further. Similarly, there is no need to further consider labels that are dominated by an existing label. Inherently, as the graph is directed and acyclic the algorithm expands the search in a breath-first manner. We note, that the list is an abstraction that can be replaced with an appropriate data-structure in order to control the expansion more precisely.

Finally, the best label (terminating at the sink vertex v_t) is returned. The best label is an invariant that is updated whenever the search comes across a label (terminating at v_t).

Processing the labels in a left-to-right order (breath-first) comes at an advantage when considering dominance as it will prune the most labels early in the search. This is an advantage as it avoids considering many labels that will be dominated later on. We note, however, that this strategy will not find feasible solutions early on. This comes at a disadvantage as such are valid upper bounds that can also be used to prune labels during the search.

4.1 Passage Points Extension

An extension is needed in order to deal with the passage point constraints since the described algorithm does not require labels to pass these locations in the pre-specified time windows.

In Algorithm 1, instead of assigning the accumulated time, $t_{l'}$, on line 11 we assign to the generated label, l', a time consumption of $\max(t_{l'}, t_p)$ where t_p corresponds to the earliest arrival time at passage point p. If no passage point exists at the considered vertex then $t_p = 0$, meaning that $\max(t_{l'}, t_p) = t_{l'}$. This modification of the algorithm keeps the structure of the problem intact, and we can use the same pruning rules. In addition, the time bound infeasibility rule is strengthened as it can be applied for the latest arrival at all passage points, not just the final destination and $T_{\max}$.

The extension has one important drawback. The produced solution may be infeasible as the modification allows a label to move forward in time without changing the position or velocity. This is naturally not possible for a train, unless it is standing still (zero velocity). We propose two different ways to accommodate this issue. Firstly, assigning a cost penalty on violations, according to the size of the violated duration would discourage solutions that arrive too early. Alternatively, a post-process can repair the infeasibility by adjusting the profile accordingly, such that it arrives later in time but at the earliest time possible at the correct speed. In the experiments we have adopted the latter.

5 Computational Results

In this section we benchmark the proposed solution method using real-life instances from various countries. A summary of the considered instances in shown in Table 1. The instances are divided into three different classes, each representing distinct trips. The first class (A) consists of intercity trips provided by a confidential railway operator. The last two classes (B and C) are intercity trips provided by the principal train operating company in Denmark (DSB). All instances include multiple speed limits and in many cases also multiple passage points. The altitude profile is not negligible, hence many piecewise linear slopes are used to approximate the track slopes. The arrival time at the final destination is given for each instance, where the input data allows the train to be delayed a few seconds.

The benchmarks are performed using a Intel (R) Core^{TM} i5-3320M CPU @ 2.60GHz processor with 8 gigabytes of main memory. The results are shown in Table 2 where the solution method described in this paper (LSA) is compared to a solver (*C-solver*) provided by Cubris[1]. LSA and *C-solver* method solve the same problem using the same objective, however, a few differences exist. First, *C-solver* does not rely on linearized track slopes. Secondly, additional soft constraints, such as minimizing the number of driver actions, are taken into account in *C-solver*. The parameter settings of *C-solver* have, however, been adjusted to make the comparison as fair as possible. For the sake of comparison, the energy consumption is normalized into watt-hours per ton-kilometer ($^{wh}/_{tk}$), which we will denote as the cost. We measure the results of LSA and *C-solver* relative to the cost of the fastest profile (*FP*). The *FP* is the speed profile that uses the minimum travel time possible for a journey. This profile follows the speed restrictions as closely as possible, which helps the train to stay ahead of time (rather than behind the schedule). This speed profile represents the choice made by an energy-inattentive driver. The cost of the *FP* highlights the potential savings of the benchmarked methods.

In general, the results show that the LSA produces high quality solutions within 0–2 seconds. We note that some of the input data can be preprocessed further, thus reducing computational time. A large portion of the time is spent in the label setting algorithm, but time spent preparing the input data and generating the underlying graph is not negligible. An average improvement of 0.3, 1.2 and 0.4 $^{wh}/_{tk}$ in cost (1.5%, 2.6% and 0.8% in savings) is observed for respectively A, B and C instances compared to the *C-solver*. Compared to the energy usage of *C-solver*, the relative improvement per kilometer is 3.6%, 4.7% and 1.5% respectively. The average improvement for all trips, including all classes, is 3.3%.

Compared to the A instances, relatively high computational times are observed for the B and C instances. This is a natural consequence of the increased distance, the number of regions and the graph size, resulting in a larger number of considered labels.

The LSA consistently arrives as late as possible, which in turn gives more room for cost reductions. In contrast to the *C-solver* method where the arrival varies around on the planned time, and even arrives before time in some cases (up to 2 seconds).

The *C-solver* finds solutions extremely fast, all measured computational times are less than 100 milliseconds. Significant gains are achieved compared to the *FP*, but the superior results of the LSA suggest that the *C-solver* sometimes makes some poor choices. We note that some discrepancy might appear in the results, since the LSA gets a linearized altitude profile as input, while the *C-solver* works on a continuous altitude profile.

Comparing the costs of the two algorithms, a few outliers need some explanation. In A07 and C06 the LSA surprisingly finds significantly worse solutions. After inspection of the results, we discovered that this is due to a combination of

Instance	Distance	Speed Limits	Passage Points	Altitude Difference	Track Slopes	Trip Time
A00	6.3	7	1	9	5	390
A01	32.9	9	5	28	13	1 020
A02	37.7	4	3	54	17	1 110
A03	29.8	7	2	48	10	930
A04	28.0	8	2	62	8	1 080
A05	19.8	12	3	43	8	780
A06	14.7	5	1	52	3	570
A07	10.9	2	1	30	4	420
A08	9.8	7	1	51	3	390
A09	7.5	3	1	11	5	300
A10	14.2	3	1	35	4	510
A11	21.3	6	3	84	3	750
A12	16.5	8	2	119	4	660
A13	22.3	7	1	105	6	720
A14	1.1	4	1	20	2	180
B00	50.3	19	10	55	27	1 575
B01	28.9	3	4	20	15	795
B02	23.5	4	2	102	10	765
B03	15.5	4	2	37	4	495
B04	14.8	6	1	19	3	465
B05	14.6	3	2	9	6	495
B06	32.8	6	4	22	11	915
B07	11.9	5	3	14	6	435
B08	18.8	9	4	26	10	765
C00	19.6	9	4	26	10	735
C01	11.9	4	3	14	6	405
C02	32.8	9	4	22	11	945
C03	14.6	5	2	9	6	465
C04	14.7	6	1	20	3	465
C05	15.5	4	2	35	4	495
C06	23.5	4	2	102	9	705
C07	28.9	3	4	19	16	795
C08	50.4	16	10	63	27	1 395

Table 1: An overview of the benchmarked real-life data instances. The columns show instance identifiers, trip distance (km), number of changing speed limits, number of passage points, maximum change in altitude (m), the number of linearized track slopes, and finally the trip time (s).

					FP	LSA			*C-solver*			
Case	Reg.	Vertices	Edges	Labels	Cost	Time	Sav	Arr.	Time	Sav	Arr.	Δ
A00	9	246	259	160	22.6	20	31.3%	-4.8	19	29.5%	1.6	1.8%
A01	26	1 122	1 606	1 490	31.9	118	16.8%	-6.7	10	13.7%	1.3	3.1%
A02	21	906	1 245	3 012	29.8	77	19.0%	-6.5	25	18.6%	1.8	0.4%
A03	17	664	866	1 237	22.6	68	17.5%	-6.3	13	17.5%	-1.0	-0.0%
A04	15	431	490	1 650	24.6	58	43.7%	5.1	3	39.3%	0.5	4.4%
A05	19	714	875	4 184	33.8	97	32.0%	-2.2	20	27.5%	1.0	4.5%
A07	5	350	488	405	27.1	15	34.9%	-3.8	84	35.6%	1.1	-0.7%
A08	3	357	511	409	33.6	15	37.1%	-5.2	36	44.3%	-0.2	-7.3%
A09	7	382	517	47	33.9	41	2.8%	-6.7	0	0.0%	-4.0	2.8%
A10	5	259	295	233	37.3	18	37.4%	-4.7	33	38.3%	0.3	-0.9%
A11	4	356	485	594	34.0	32	29.9%	-4.6	31	30.0%	0.0	-0.2%
A12	7	629	1 049	695	20.1	57	32.3%	-1.3	17	32.2%	-1.9	0.1%
A13	10	392	491	1 106	39.4	40	17.9%	-5.4	23	16.9%	-0.6	1.0%
A14	11	572	736	378	18.0	41	50.8%	-2.8	84	49.3%	1.5	1.5%
A16	3	43	40	54	12.6	6	83.8%	45.1	7	71.8%	-0.7	12.0%
Avg							32.5%			31.0%		1.5%
B00	61	5 756	8 900	112 756	46.2	1 248	50.6%	-4.5	23	38.9%	0.3	11.7%
B01	22	3 124	5 169	55 264	43.2	499	36.3%	-5.2	23	35.5%	0.9	0.8%
B02	14	2 156	3 310	60 223	44.0	426	46.1%	-5.3	86	41.4%	1.1	4.7%
B03	7	1 579	2 562	11 110	56.4	130	44.2%	-5.4	38	42.6%	1.8	1.6%
B04	7	1 040	1 508	4 181	52.3	49	42.8%	-4.0	27	42.4%	0.4	0.4%
B05	9	1 985	3 241	22 633	51.0	183	55.3%	-5.6	42	53.2%	-0.8	2.1%
B06	21	3 492	5 999	84 476	42.9	635	37.8%	-4.7	41	36.6%	-1.1	1.2%
B07	13	1 930	3 067	12 277	52.8	143	60.3%	-5.0	46	59.9%	-1.5	0.4%
B08	23	2 039	2 873	42 344	47.5	314	71.0%	-5.0	64	70.3%	1.7	0.7%
Avg							49.4%			46.7%		2.6%
C00	23	2 053	3 100	22 360	49.3	208	53.1%	-4.8	54	52.3%	-1.0	0.9%
C01	12	2 023	3 303	10 577	59.6	201	50.2%	-5.2	37	49.5%	-0.8	0.7%
C02	24	2 738	4 159	25 868	45.5	342	41.1%	-4.5	24	37.4%	-0.7	3.7%
C03	12	1 921	3 028	13 074	52.4	163	47.3%	-5.3	33	45.2%	1.7	2.0%
C04	7	1 088	1 608	2 854	50.3	51	44.3%	-4.6	30	43.3%	1.7	1.0%
C05	7	2 014	3 504	26 599	46.3	220	55.3%	-6.4	29	53.4%	1.3	1.9%
C06	13	1 643	2 404	6 311	44.9	160	35.0%	-5.6	22	39.5%	-1.5	-4.5%
C07	25	2 857	4 434	35 412	44.1	395	34.6%	-5.3	41	33.7%	0.9	0.9%
C08	57	5 842	9 356	93 436	38.1	1 143	35.7%	-5.7	36	35.2%	1.5	0.6%
Avg							44.1%			43.3%		0.8%

Table 2: An overview of the benchmark results. The first columns show the instance identifier, number of regions, vertices and edges in the underlying graph. The forth column shows the number of generated labels, and the fifth shows the energy cost for the fastest profile (*FP*). For each method we report runtime (ms), savings (relative to *FP*) and the remaining trip time (s). The final column shows the difference in savings.

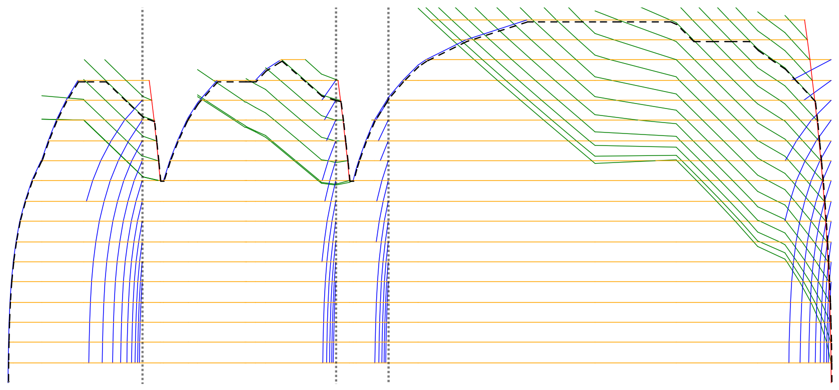

Figure 5: An illustration of an optimized speed profile in the graph presented in Section 3.2. The profile, depicted as a dashed black line, follows a series of accelerate-cruise-coast-brake stages. Multiple acceleration and braking curves are present due to speed restrictions. Many acceleration curves are inserted before passage points (vertical dotted lines) to avoid (potential) infeasibility.

large segments in the altitude profile and the discrete set of cruising speeds. This considerably restricts the choices of speed for LSA. Splitting the long segments in the altitude profile into smaller parts will reduce this problem.

A relative high difference is observed in a few cases: A05, A16, B00, B02 and C02. After inspecting these cases more carefully we conclude that the solutions are correct. The differences are a result of greedy heuristic choices when facing speed limits and passage points.

An example of a found optimized speed profile using LSA is illustrated in Figure 5. In the example the passage points are not very binding, however it can be observed how the solution method distributes surplus time by letting the train coast in three different segments. A second example is shown in Figure 6. In this example it is observed that the passage point is now more binding. An excess of surplus time is available before the first passage point, however, in order to avoid infeasibility after this point a high speed is required in the sequel.

Comparing Figure 5 and 6, it is seen that if the passage points are not binding, the algorithm strives towards a constant speed during the whole trip (except when speed limits are imposed). If some passage points are binding, the optimal speed profile may encompass low speed up to the passage point, and a much higher speed in the last part.

We demonstrated that increasing the number of partial coasting profiles in the underlying graph of the LSA method can improve the solutions slightly in many cases, and significantly in few cases. The latter can be solved by refining the algorithm to identify such corner cases - note that these instances are still solved very quickly. The experiments suggest that the LSA method is able to find very good solutions quickly. Inserting additional coasting profiles is questionable, as the discretization reaches a limit where a driver is unable to

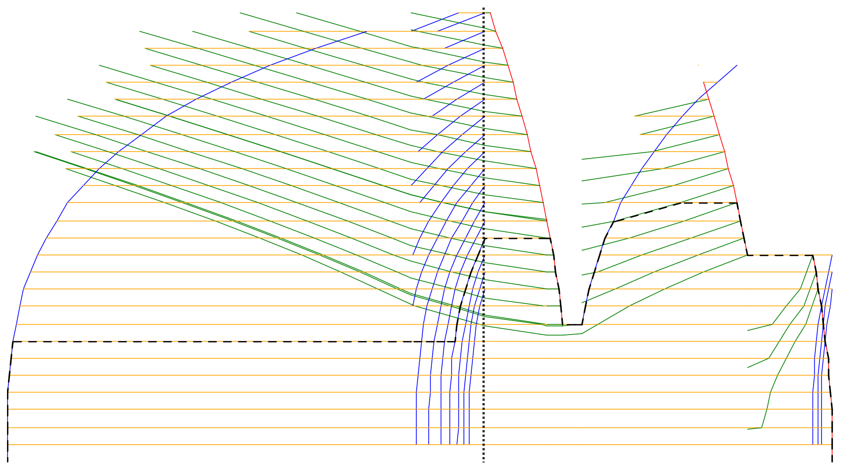

Figure 6: An illustration of an optimized speed profile in the graph presented in Section 3.2. The profile, depicted as a dashed black line, follows a series of accelerate-cruise-coast-brake stages. Multiple acceleration and braking curves are present due to speed restrictions. Many acceleration curves are inserted before passage points (vertical dotted lines) to avoid (potential) infeasibility.

react accordingly.

6 Better bounds

The LSA solution method is heuristic as only a subset of the partial profiles are generated in the speed profile graph. However, we expect to find high quality solutions since all necessary partial profiles for acceleration, braking and cruising exist for obtaining an optimal solution. Only the possible coasting opportunities are constructed from a discrete set of points where the train can start coasting.

We therefore conduct an additional experiment with a considerably increased number of coasting opportunities. We have added coasting profiles that will hit the braking curve at every discrete speed step (0 km/h, 1 km/h, 2 km/h ...). One cannot expect the driver to navigate with a larger precision so the found solutions provide a lower bound for how good the algorithm can perform.

The results are listed in Table 3. An average improvement of 1.0% is observed, where more than half of the improvement solely stems from 5 instances (A0, A04, A08, A10, A14). As expected, the required cpu-time and the number of vertices, edges and labels generated increases. In most cases, the improvement is less than 0.5% indicating that LSA is obtaining solutions very close to the optimal solution. Only in 5 instances we observe results above 1.5%. By close inspection it can be seen that there exist corner cases where the generation of the coasting profiles is too crude in the LSA. A comparison of the solutions found in A04 is illustrated in Figure 7.

Case	Time	Improvement	Vertices	Edges	Labels
A00	23	3.1%	214.63%	213.51%	247.50%
A01	476	0.0%	143.67%	132.50%	1902.75%
A02	142	0.2%	150.66%	149.00%	379.85%
A03	51	0.9%	120.33%	113.63%	249.88%
A04	88	4.1%	152.67%	143.67%	466.55%
A05	170	0.4%	190.34%	196.11%	392.97%
A07	40	1.4%	139.43%	98.36%	773.33%
A08	45	11.3%	264.99%	268.10%	448.17%
A09	20	1.3%	203.47%	194.92%	257.08%
A10	47	6.9%	232.02%	226.39%	590.24%
A11	38	0.2%	81.88%	48.05%	281.29%
A12	58	0.1%	179.85%	168.84%	189.15%
A13	46	0.1%	183.39%	195.11%	595.50%
A14	27	6.3%	190.25%	133.91%	184.37%
A16	4	0.0%	130.23%	77.50%	61.11%
B00	4.463	0.2%	90.36%	82.82%	226.00%
B01	1.992	0.0%	82.23%	76.51%	263.51%
B02	2.000	0.2%	136.36%	111.00%	251.60%
B03	5.107	0.5%	213.93%	212.45%	3315.22%
B04	145	0.2%	226.35%	235.08%	210.36%
B05	8.036	0.3%	165.74%	156.87%	2388.19%
B06	3.560	0.0%	105.07%	101.65%	308.68%
B07	380	0.0%	131.61%	114.84%	229.70%
B08	1.139	0.5%	151.94%	148.42%	188.97%
C00	926	0.2%	156.99%	160.06%	367.53%
C01	408	0.0%	151.85%	146.35%	287.17%
C02	814	1.2%	123.05%	111.97%	272.39%
C03	5.566	0.4%	174.28%	169.15%	3133.80%
C04	130	0.4%	196.42%	197.70%	265.98%
C05	769	1.0%	99.06%	85.16%	250.36%
C06	226	0.3%	184.42%	187.94%	219.39%
C07	1.207	0.1%	122.37%	119.28%	277.94%
C08	48.292	0.2%	74.14%	65.26%	2367.23%

Table 3: Overview of results obtained by increasing the number of coasting partial profiles. The columns respectively show the instance identifier, runtime in milliseconds, relative improvement in energy savings and relative increase in the number of vertices, edges and labels.

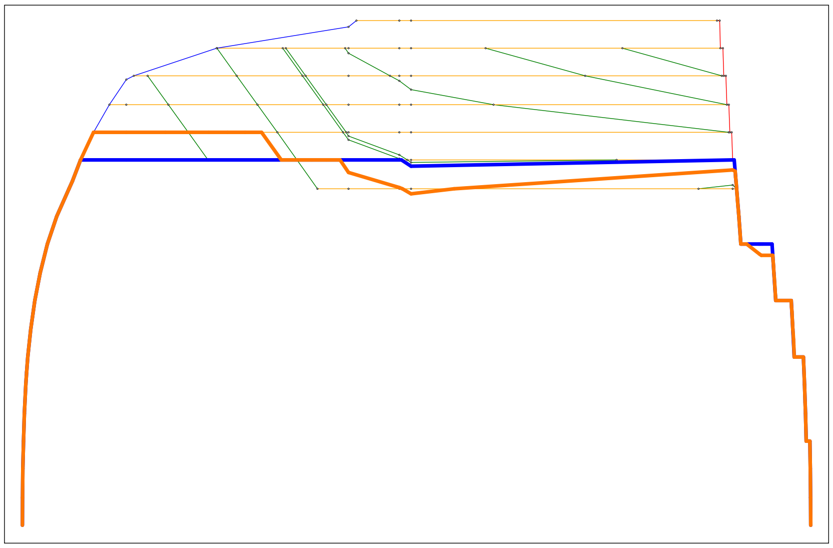

Figure 7: The plot compares two solutions. The thick blue path shows the solution found using the LSA method. The thick yellow path shows the solution found using the same method with an increased number of partial coasting profiles. The solutions are plotted on the original graph. The coasting profile discretization is too coarse to generate a good solution for the LSA method.

The total cpu-time required in the last experiment clearly violates the targeted real-time requirements, however in most cases the cpu-time is less than one second. We therefore suggest a hybrid option to achieve an improved hybrid variant of the LSA method. First, the observed corner-cases can be improved as discussed. Second, a coarse discretization can be used to obtain an initial high quality solution. The initial solution can then be used as guidance in a next iteration, as we now know approximately when the train should start coasting. Furthermore, an initial solution can help speed up the labeling algorithm by pruning relevant branches early in the search-process.

7 Further extensions

The developed framework is very flexible and can easily be extended to handle a number of extra criteria.

First of all, we notice that LSA not only returns the optimal solution, but actually for the sink v_t returns a set of Pareto-optimal labels $(cost(l), time(l))$, satisfying $time(l) \leq T_{\max}$. Each of these labels represents a feasible speed profile, and can be evaluated with respect to various soft constraints (e.g. robustness, passenger comfort, environmental impact from noise and vibration) in order to choose the overall best profile.

It is also easy to handle extra criteria explicitly in the solution approach. For instance, the drivers would like to have profiles without too many changes in operation (power-hold-coast-brake). For this purpose each label l can be extended to three parameters $(cost(l), time(l), changes(l))$ where $changes(l)$ denotes the number of speed changes. In the label setting algorithm $changes(l)$ is increased by one each time an edge corresponds to a change in operation. The dominance rule in Theorem 3 is easily adapted to handle the new parameter.

Finally, robustness of a speed profile can be handled by using the techniques from [30]: For every arc $a \in A$ assume that there is a probability p_a for disruption. For instance an arrival to a passage point at the extreme ends of the corresponding time window will have a large probability for disruption. Similarly, segments traversed close to the speed limit do not leave much space for recovery and will have a large probability p_a. If a given speed profile consists of arcs $a \in P$ we want to maximize the probability of no disruptions, hence having a second criteria to optimize

$$\max \prod_{a \in P} (1 - p_a)$$

Since the log function is monotone this corresponds to maximizing

$$\max \log \prod_{a \in P} (1 - p_a) = \max \sum_{a \in P} \log(1 - p_a) = \min \sum_{a \in P} - \log(1 - p_a),$$

so by choosing edge weights $w_a = -\log(1 - p_a)$ we can easily handle the robustness aspect by using labels $(cost(l), time(l), prob(l))$, where $prob(l)$ is the sum of the weights w_a on the trip. As before, an upper limit on the probability of disruption can be assigned to each trip when selecting the most energy efficient speed profile.

8 Conclusion

Optimizing train speed profiles is important research as trains are one of the major electricity consumers in most countries. Minimizing even a small percentage leads to significant savings and reduces the environmental impact. A variety of solution methods exist in literature but further improvements can be made in terms of computational requirements and increased realism or accuracy.

We proposed a novel DP-based solution method for finding optimized speed profiles. Although it is not an exact method for finding the optimal solution, our computation results show that very high quality solutions are indeed found within 1-2 seconds in the worst case. The computational time savings are significant compared to previous methods based on DP. A large number of real-life instances have been considered. A comparison to *C-solver*, an existing commercial method, shows that the solution quality is high. We conclude that the proposed solution method is suitable for providing real-time driving assistance.

The solution framework is elegant and flexible. The underlying graph can be extended without the use of advanced mathematics. The solver is interchangeable as the RCSPP can be solved using different methods such as Lagrangian relaxation, constraint programming or even heuristic methods.

For future research, a number of interesting extensions are worth investigating. The solution method could be extended to handle steep gradients where cruising is not possible, and extended to consider trains with discrete speed control. The potential of the presented methodology is not fully exploited. Even more aggressive preprocessing methods could improve computational time as well as tuning of the label setting algorithm. Finally, we believe that proposed framework is eligible for other types of transportation modes, examples include trams, trucks, air-planes and vessels.

References

[1] Cubris aps. `http://http://www.cubris.dk/`. Accessed: 2015-06-25.

[2] Innovative integrated energy efficiency solutions for railway rolling stock, rail infrastructure and train operation. `http://www.railenergy.org`. Accessed: 2015-06-26.

[3] Amie Albrecht, Phil Howlett, Peter Pudney, Xuan Vu, and Peng Zhou. The key principles of optimal train control—part 1: Formulation of the model, strategies of optimal type, evolutionary lines, location of optimal switching points. *Transportation Research Part B: Methodological*, pages –, 2015.

[4] Amie Albrecht, Phil Howlett, Peter Pudney, Xuan Vu, and Peng Zhou. The key principles of optimal train control—part 2: Existence of an optimal strategy, the local energy minimization principle, uniqueness, computational techniques. *Transportation Research Part B: Methodological*, pages –, 2015.

[5] Amie R. Albrecht, Phil G. Howlett, Peter J. Pudney, and Xuan Vu. Energy-efficient train control: From local convexity to global optimization and uniqueness. *Automatica*, 49(10):3072 – 3078, 2013.

[6] A.R. Albrecht, P.G. Howlett, P.J. Pudney, X. Vu, and P. Zhou. Energy-efficient train control: the two-train separation problem on level track. *Journal of Rail Transport Planning & Management*, 5:163–182, 2015.

[7] T. Albrecht. The influence of anticipating train driving on the dispatching process in railway conflict situations. *Networks and Spatial Economics*, 9(1):85–101, 2009.

[8] I.A. Asnis, A.V. Dmitruk, and N.P. Osmolovskii. Solution of the problem of the energetically optimal control of the motion of a train by the maximum principle. *USSR Computational Mathematics and Mathematical Physics*, 25(6):37 – 44, 1985.

[9] C.S. Chang and S.S. Sim. Optimising train movements through coast control using genetic algorithms. *IEE Proceedings - Electric Power Applications*, 144(1):65–73, Jan 1997.

[10] Jiaxing Cheng. *Analysis of optimal driving strategies for train control problems.* University of South Australia, 1997.

[11] Jiaxing Cheng, Yelena Davydova, Phil Howlett, and Peter Pudney. Optimal driving strategies for a train journey with non-zero track gradient and speed limits. *IMA Journal of Management Mathematics*, 10(2):89–115, 1999.

[12] Andrea D'Ariano, Marco Pranzo, Ingo Hansen, et al. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *Intelligent Transportation Systems, IEEE Transactions on*, 8(2):208–222, 2007.

[13] CW Davies, Ion Association, et al. Butterworths, 1962.

[14] Rudiger Franke, Peter Terwiesch, and Markus Meyer. An algorithm for the optimal control of the driving of trains. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 3, pages 2123–2128. IEEE, 2000.

[15] Seong-Ho Han, Yun Sub Byen, Jong Hyen Baek, Tae Ki An, Su-Gil Lee, and Hyun Jun Park. An optimal automatic train operation (ato) control using genetic algorithms (ga). In *TENCON 99. Proceedings of the IEEE Region 10 Conference*, volume 1, pages 360–362 vol.1, 1999.

[16] I.A. Hansen and J. Pachl. *Railway Timetabling & Operations.* Eurailpress, Hamburg, Germany, 2014.

[17] Phil Howlett. An optimal strategy for the control of a train. *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, 31(04):454–471, 1990.

[18] Phil Howlett. The optimal control of a train. *Annals of Operations Research*, 98(1-4):65–87, 2000.

[19] Phil G Howlett, Peter J Pudney, and Xuan Vu. Local energy minimization in optimal train control. *Automatica*, 45(11):2692–2698, 2009.

[20] Philip G Howlett and Peter J Pudney. *Energy-efficient train control.* Springer, 1995.

[21] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and MariusM. Solomon, editors, *Column Generation*, pages 33–65. Springer US, 2005.

[22] Jyh-Cherng JONG and Sloan CHANG. Algorithms for generating train speed profiles. *Journal of the Eastern Asia Society for Transportation Studies*, 6:356–371, 2005.

[23] E. Khmelnitsky. On an optimal control problem of train operation. *Automatic Control, IEEE Transactions on*, 45(7):1257–1266, Jul 2000.

[24] H Ko, T Koseki, and M Miyatake. Application of dynamic programming to the optimization of the running profile of a train. In *Computers in railways IX*, 2004.

[25] Maialen Larranaga, Jonatha Anselmi, Urtzi Ayesta, Peter Jacko, and Asier Romo. Optimization techniques applied to railway systems. Technical report, Basque Center for Applied Mathematics, January 2013. 17p.

[26] Xiang Li and Hong K. Lo. An energy-efficient scheduling and speed control approach for metro rail operations. *Transportation Research Part B: Methodological*, 64:73 – 89, 2014.

[27] Rongfang Rachel Liu and Iakov M Golovitcher. Energy-efficient operation of rail vehicles. *Transportation Research Part A: Policy and Practice*, 37(10):917–932, 2003.

[28] Shaofeng Lu. *Optimising power management strategies for railway traction systems*. PhD thesis, University of Birmingham, 2011.

[29] D. Pacciarelli and M. Pranzo. Speed regulation in rail networks. In *Proceedings of the 11th IFAC Symposium on Control in Transportation Systems (CTS 2006)*. Delft, The Netherlands, 2006.

[30] Line Blander Reinhardt and David Pisinger. Multi-objective and multi-constrained non-additive shortest path problems. *Computers & Operations Research*, 38(3):605–616, 2011.

[31] Horst Strobel, Peter Horn, and Manfred Kosemund. Contribution to optimum computer-aided control of train operation. In *Proc. 2nd IFAC/IFIP/IFORS Symp. Traffic Control and Transportation Systems, Monte-Carlo*, pages 377–387, 1974.

[32] Y. Wang, B. De Schutter, T.J.J. van den Boom, and B. Ning. Optimal trajectory planning for trains – a pseudospectral method and a mixed integer linear programming approach. *Transportation Research Part C: Emerging Technologies*, 29:97–114, 2013.

[33] Yihui Wang, Bing Ning, Fang Cao, Bart De Schutter, and Ton JJ Van den Boom. A survey on optimal trajectory planning for train operations. In *Service Operations, Logistics, and Informatics (SOLI), 2011 IEEE International Conference on*, pages 589–594. IEEE, 2011.

[34] Liu Wei, Li Qunzhan, and Tang Bing. Energy saving train control for urban railway train with multi-population genetic algorithm. In *Information Technology and Applications, 2009. IFITA'09. International Forum on*, volume 2, pages 58–62. IEEE, 2009.

[35] KK Wong and TK Ho. Coast control for mass rapid transit railways with searching methods. *IEE Proceedings-Electric Power Applications*, 151(3):365–376, 2004.

[36] Xin Yang, Xiang Li, Bin Ning, and Tao Tang. A survey on energy-efficient train operation for urban rail transit. *Intelligent Transportation Systems, IEEE Transactions on*, 17(1):2–13, Jan 2016.