# A Self-Bounding Branch & Bound Procedure for Truck Routing and Scheduling

Csongor Gy. Csehi
Budapest University of Technology and Economics, Műegyetem rkp. 3-9, Budapest, Hungary
E-mail: cscsgy@cs.bme.hu

Ádám Tóth, Márk Farkas
Nexogen Ltd. Alkotás u. 53, Budapest, Hungary
E-mail: toth.adam@nexogen.hu, mark.farkas@nexogen.hu and nexogen.com

**Keywords:** logistics, route optimization, branch and bound

**Received:** October 30, 2018

*In this paper we study a part of a core algorithm of a complex software solution for truck itinerary construction for one of the largest public road transportation companies in the EU. The problem is to construct a cost optimal itinerary, given an initial location with an asset state, the location and other properties of tasks to be performed. Such an itinerary specifies the location and activity of the truck and the driver until the completition of the last routing task. The calculation of possible itineraries is a branch and bound algorithm. The nodes of the search tree have the following arguments: position, time, driver-state and truck-state. For each node we calculate the cumulative cost for the road reaching that state, and a heuristic lower bound for the cost of the remaining road. In each step the procedure expands the next unexpanded node with the best sum for cumulative and heuristical costs.*
*It is hard to give a sharp lower bound if the model contains time windows. To make a sharp heuristic we run the same branch and bound algorithm (from each node) but with simplified data (hypothetical positions and simplified activities: no refuelling, no road costs, etc.). We have reached significant gains in performance and quality compared to the previous approach.*

*Povzetek: Članek predstavlja aplikacijo za razporejanje tovornjakov v enem največjih transportnih podjetij v EU. Za reševanje problema je uporabljena metoda razveji in omeji. Da bi izboljšali hevristiko, je bila uporabljena ista razveji in omeji metoda iz vsakega vozlišča na poenostavljenih podatkih. Končni rezultat je pomembno izboljšan v primerjavi s predhodnimi pristopi.*

## 1 Introduction

We will study a part of a core algorithm of a complex software solution for truck itinerary construction for one of the largest public road transportation companies in the EU. A minor improvement on the operational cost of each tour can result huge advantage for the freight services company.
The problem is to construct a cost optimal itinerary, given an initial location with an asset state, the location and other properties of tasks (we will call them routing tasks) to be performed. Such an itinerary specifies the location and activity of the truck and the driver until the completition of the last routing task. This means that this itinerary gives every instruction to the driver, including every turn in the road and every stops with exact durations, etc. The working stops can be done only in the places of the tasks, the refueling and resting stops can be done only in previously fixed places (roughly 4000 fixed parking places and 100 fixed filling stations across Europe). To achieve such an itinerary we use mapping software to construct the routes and calculate the distance, duration and cost between any two locations. Clearly the problem is much harder than a path finding in the graph, because we can do many different actions in each place (different amount of fuel taken, different duration of rest, etc).
The software (which also performs the vehicle assignment) is already finished and applied with positive results (from 2015), large cost saving is achieved by the company. For more formal definitions of the problem, and more information of the software one must read [4]. The ongoing researches aim to extend the functionality of the software. One goal is to improve optimality by planning the itinerary for longer timespan. That means more routing tasks have to be calculated each round.

## 2 History

This area of operations research is widely studied [1]. However, to build a complex solver for truck routing and scheduling is a novel concept. This is the reason why we have collected some approaches and solutions that are simi-

lar in methodology and solution concept not in the problem in this section. For detailed information on the literature of the problem one can read [4].
There is no such concept in the literature which contains the idea of using the same algorithm on simplified data to get the lower bounds for a branch and bound method. Hence, we have named our method Self-Bounding Branch and Bound (SBBB).
There are many approaches using so called Recursive Branch and Bound (RBB) methods [7]. However, they obtain the bounds by a recursive strategies using the information from preceding calculations [8]. In SBBB we do not use any previously calculated information during the lower bound calculation. In our particular problem the driver state makes this approach impossible because the state is different in all the nodes of the searching tree. In some cases (such as in rectangular guillotine strip packing problem [5]) the recursivity can be formalized in functions which help the calculation of the bounds.
Searching for SBBB alike concepts one can find the Fractal Branch and Bound (FBB) [9]. However, FBB is a very special method where the searching field is self-similar. This means that FBB is more likely an RBB because it uses simplification on the often calculated values. That way, it is different from SBBB and it is not applicable to our problem because of the different driver states of the nodes.
The Double Branch and Bound (DBB) methods are algorithms where the lower bound for a branch and bound method is calculated by another branch and bound method [11], [10]. This means that SBBB is a special type of DBB. However, the DBB concept can be applied for small instances in reasonable timeframe (see some computational results in [10]). It is usual to use RBB like methods for $m$-machine permutation flowshop problems [2]. Moreover there are some approaches where they apply DBB methods [3] instead of the recursive functions.
As we mentioned before no such concept can be found in the literature where the same algorithm on simplified data gives the lower bound of a branch and bound method. However, our results (see Section 4) shows that for some problems with proper reconciliation SBBB can solve large problem instances (larger than DBB).

# 3 The algorithm

The calculation of possible itineraries is a branch and bound algorithm. For detailed information on the widely used algorithms of operations research the reader should see [1] The nodes of the search tree have the following arguments: position, time, driver-state and truck-state (we will call these data the state). For each node we calculate the cumulated cost for the itinerary reaching that state, and a heuristical lower bound for the cost of the remaining road. Each node has a pointer to its parent (this will make it possible to calculate the roue from a proper node). In each step the procedure expands the next unexpanded node with the best sum for cumulated and heuristical cost.
The following oversimplified example of [4] with Figure 1 illustrates the tree of the algorithm. Suppose that we are in position 'Start' in the begining. From 'Start' we can go to different places for example two parking places 'P1' and 'P2' (the state will be different in the two locations if the duration and distance of the drivings are not equal). Supposing that we can rest 9 or 11 hours we get two new nodes from each parking place reaching node. If we can reach place 'P3' from both 'P1' and'P2' then this way we get four different nodes in the same place 'P3'. In general none of the four nodes can be bounded in the algorithm, because the states are different and hence, we can not predict which will give the best solution in the end.

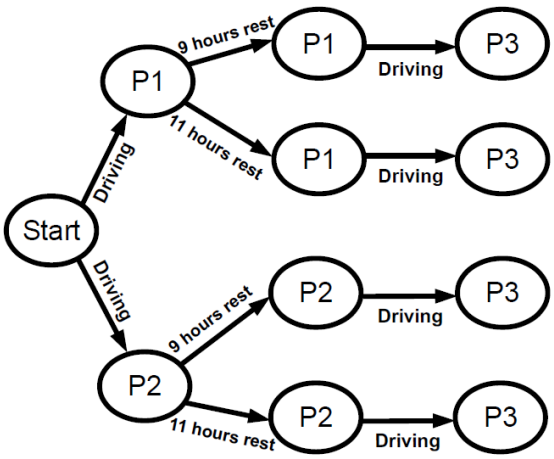


Figure 1: An example subtree of the algorithm

The algorithm has many additional logics, but here we focus on the heuristics only. A more detailed description of the algorithm can be found in [4].
The better the lower bounds are, the less nodes need to be expanded. However, it is always more time-consuming to make better estimations. The difficulty with the heuristics is the presence of the state of the driver (driver state) and the opening times of the routing tasks (time window). Both would be easier to handle separately but together it gives an NP-hard problem. The current solution optimizes the routes for each task in a tour separately, because the current heuristics are too weak and that way the algorithm would be too slow (would expand too many nodes).
The main steps of the algorithm are the following:

**Algorithm 1.** *1. Create the starting node of the tree from the initial state and position of the driver. Put it in an empty list $L$. Set the upper bound $U = \infty$.*

*2. While $L$ has any element:*

*(a) Pick $X$ from $L$ with the best TotalCost value.*

*(b) If the itinerary given by $X$ is a complete tour (finishing all the routing tasks), then RETURN $X$.*

*(c) Select the best possible activities (set $A$) to do from $X$.*

*(d) BRANCHING: For each element of $A$ create the node (set $N$) which represents the state and position after that activity.*

*(e) For each element of $N$ calculate the cummulated cost (we can get it by adding the cost of the activity to the cummulated cost of $X$).*

*(f) BOUNDING 1: For each element of $N$ calculate the lower bound for the remaining cost (this will be examined in detail in the next sections). Discard the nodes from $N$ which has a total cost (it is the sum of the cummulated and the remaining costs) higher than the upper bound $U$.*

*(g) BOUNDING 2: For each element of $N$ compare the lower bound for the reaching time with the limitations (we get the lower bound during the calculation of the heuristics). If the node can not reach the target in time than delete it from $N$.*

*(h) BOUNDING 3: For each element $Y$ of $N$, where the place of $Y$ is $P$, get the list $L_P$ of the previously examined nodes in place $P$. Compare $Y$ with every element of $L_P$, and if there exists such $Z$ that every state related variable and the cost are not worse in $Z$ than in $Y$, then delete $Y$ from $N$.*

*(i) For each element of $N$ put it into $L$ and into the proper $L_{P_i}$ list according to the place of the nodes.*

*(j) For each element $C$ of $N$ which is a complete tour, let the upper bound $U = min(U, cost(C))$.*

3. *RETURN: Unreachable target. The target can not be reached in the given time limit.*

The first bounding is the usual bounding of BB methods. The second bounding discards the infeasible solutions. The third bounding is a dominance criteria which uses dominance relation between partial solutions.
In this paper a solution is given for the efficient lower bound calculation (in Step 2/f). This will be done by a similar branch and bound structure but with simplified data (relaxed field of POIs).

## 3.1 The concept of the main bounding method

As we mentioned in step 2/f of the algorithm we need a good heuristic to bound the remaining cost from every node. For this we need to calculate a minimal itinerary and we have to bound the needed duration.
First we estimate the remaining distance and driving duration. To optimize the running time we do not want to make the mapping software calculate all these estimations, but store as much of the possibly needed information as we can. We construct two graphs where the nodes are the possible POIs of the tours (parking places, filling stations, ferries, tunnels, etc.) and the length of the edges are the minimal distances and driving durations (we get those values using PTV Group softwares). From these graphs we generate the minimal distances and durations between each pair of nodes with the Floyd – Warshall algorithm (Floyd [6] 1962; Warshall [12] 1962). This happens in a precalculation phase before the itinerary generator algorithm. It is a separate topic how we handle the truck positions and places of the routing tasks (since they are not permanent, hence, they are not contained in the graphs).
These graphs can be used during the algorithm instead of PTV, thus accelerating the computation that way.
After we have a lower bound for the remaining driving time we estimate the total time needed by constructing a hypothetical itinerary. These are the steps of the estimation:

**Algorithm 2.** *1. Let $X$ be the remaining driving time needed to reach the target. Let $D$ be the state of the driver, and $T$ = the actual time.*

2. *$Y$ = the amount the driver with driver-state $D$ can drive continuously.*
3. *Let $Z = min\{X, Y\}$.*
4. *$D$ = the driver-state after a $Z$ long drive.*
5. *$X = X - Z$; $T = T + Z$.*
6. *If $X = 0$ then RETURN $T$.*
7. *$R$ = the amount what the driver with driver-state $D$ should rest.*

We suppose that the driver can drive the maximal available driving time, each time, and then reaches a parking place. In each parking place the driver rests the minimal amount what is needed and then proceeds further. When the driver reaches a routing task sometimes he has to wait for the time-window. However, supposing that there are no time-windows the heuristics can be calculated in linear time (we will call it linear heuristic).
On the other hand, if we think about how to include the time-windows in the linear heuristic we face a problem. Namely, sometimes it would be better to rest more, not just the minimal needed amount before making the task. The following example highlights that behavior.
Suppose that the driver arrives at 6 a clock, after 9 hours of driving, but the routing task opens at 10 a clock. To finish the routing task, the driver has to work 1 hour there and we have one more routing task which is 2 hours far from this. When will we finish the last routing task?
1. If we wait for the first opening and work 1 hour, then we cannot drive further because of the daily driving time limit (9 hours). That way we have to rest at least 9 hours. After the rest we can drive to the next routing task and finish it until **23 a clock**.
2. If we rest 9 hours instead of the 4 hours waiting then

we can start the work with a fresh state, drive to the next routing task and finish it until **19 a clock**.
The above example shows that we can not make good lower bounds with Algorithm 2 (linear heuristic) if we try to optimize with the driver-state and the time-windows at the same time. However, to obtain better estimations for the branch and bound procedure we must include the time-windows in the heuristics. For the best fit (between the heuristics and the algorithm) we apply the same logics to calculate a lower bound for the duration as we use in the branch and bound algorithm itself (see Section 3.2), but relax the field of POIs.
With the linear heuristic we can not efficiently optimize the tours with more than one tasks together. That way the current solution runs Algorithm 1 for each task in a tour separately (after each other, because they use the finishing driver state of the previous one).

## 3.2 The self-bounding branch and bound algorithm

As we mentioned before the main branch and bound algorithm works on nodes with position, time, driver-state and truck-state. The positions are real locations on the map. We present the algorithm B&B heuristic below, to make a sharp heuristic in step 2/f of the original algorithm. Mainly we run the same branch and bound algorithm (from each node) but with hypothetical positions (and simplified activities: no refuelling, no road costs, etc.).

**Algorithm 3.** *1. Create the starting node $S'$ of the inner searching tree. $S'$ has the same state and position of the driver as the node in the outer branch and bound (to which this algorithm results the lower bound).*

*2. Put $S'$ in an empty list $L'$.*

*3. While $L'$ has any element:*

*(a) Pick $X'$ from $L'$ with the best TotalCost value.*

*(b) If the itinerary given by $X'$ is a complete tour (finishing all the routing tasks), then RETURN $X'$.*

*(c) Select the best possible hypothetical activities (set $A'$) to do from $X'$.*

*(d) For each element of $A'$ create the node (set $N'$) which represents the state and position after that activity.*

*(e) For each element of $N'$ calculate the cummulative duration (we can get it by adding the duration of the activity to the cummulative duration of $X'$).*

*(f) For each element of $N'$ calculate the heuristical duration (with the linear heuristics).*

*(g) For each element of $N'$ compare the lower bound for the reaching time with the limitations. If the node can not reach the target in time than delete it from $N'$.*

*(h) For each element $Y'$ of $N'$, where the place of $Y'$ is $P'$, get the list $L'_P$ of the previously examined nodes in place $P'$. Compare $Y'$ with every element of $L'_P$, and if there exists such $Z'$ that every state related variable and the cost are not worse in $Z'$ than in $Y'$, then delete $Y'$ from $N'$.*

*(i) For each element of $N'$ put it into $L'$ and into the proper $L'_{P_i}$ list according to the place of the nodes.*

*4. RETURN: Unreachable target. The target can not be reached in the given time limit.*

This algorithm is similar to the linear heuristic from the aspect, that it generates those positions which was used by the linear heuristic. However, this algorithm lets the different cases compete in total duration. The best solution will give the B&B heuristic, that will be the lower bound for the remaining cost of the node in Algorithm 1 (the main branch and bound algorithm).
Observe that the B&B heuristic needs a lower bound too. For this we can use Algorithm 2 (the linear heuristic).
The real difference with Algorithm 1 is in Step 3/c. Here we generate many hypothetical parking places on the fastest road between $S'$ and the goal. Each time Algorithm 3 reaches Step 3/c new parking places can be generated.
It is easy to see that this extended procedure can give much better lower bounds for the main branch and bound algorithm, but it is questionable that if it is worth the extra time required to construct the nodes (calculating their heuristic values). Observe that it is more likely to get better heuristics this way if we have more routing tasks (with time-windows).

### 3.2.1 Driver state penalty

The objective function includes not just the real costs of the itineraries but the cost of the driver's work, the amortization of the car and many other things. The hardest part is the evaluation of the driver's state in the end of the tour. Of course it is better if the driver can drive more in the day after finishing the last working task because that way the location of the next task can be approached earlier. In fact every variable of the driver-state can be important in the end of the tour. We mainly apply a highly tested linear combination of these quantities.
The original lower bound (Algorithm 2) could not include this type of cost. Algorithm 2 gives a hypothetical fastest solution, this way it can be the case that the driver's state is much worse than in a slower solution, hence, it would not be a lower bound if we add the penalty.
However, Algorithm 3 can also handle the driver state penalty, because if those costs are included to the objective function of the inner branch and bound method, then it will lead to the best hypothetical solution which will give a lower bound on the best possible solution's objective function. It is not trivial why this approach gives a lower bound.

It mainly depends on the fact that the finishing driver state's penalty cannot be higher if we put a parking later. This paper does not aim to prove this fact.
This opportunity is very important because the driver state penalty can give about one third of the total cost of a tour. This means that compared to our previous lower bounds this approach can be much more efficient. Hence, if we include the driver state penalty to the heuristics the searching tree of the main algorithm will be much thinner.

#### 3.2.2 Longer tours

Applying Algorithm 3 we could extend the optimization to tours (with $2-3$ tasks) not just separated tasks. This means that we run Algorithm 1 with more than one targets at once, and in step 2/f we run Algorithm 3. Observe that this means that the lower bound calculation (step 2/f) can get more than one tasks at a time also.
The algorithm was a success in practice. Not just that it could plan a longer time period for a truck, but also gives more profitable plans. Building a route in a chain like concept (such as our original solution) optimizes for mid-route objective functions also and fix the earlier parts of the route in each chain. That way it was anticipated that using the new algorithm we could get better objective function values in the end of the tours. Indeed the new algorithm have given better results than the original.
A detailed description of the performance of the new algorithm can be found in Section 4.

## 4 Results

We evaluated the differences using a sample pack of $4500$ long tours. In average, these tours contains $2-3$ routing tasks with time windows. We have applied Algorithm 1 with the linear heuristic (Algorithm 2) for all tasks of those tours separately. Then we have applied Algorithm 1 with the new heuristic (Algorithm 3) for the same tours, but without decomposing them into tasks.

### 4.1 Sizes and speed

We use about $100$ parallel machines. Hence, the software runs in about 20 minutes to construct the $4400$ itineraries. However, here we give the performance data in total (summarized for the parallel machines).
The original branch and bound procedure created $3.42 \cdot 10^4$ nodes, inserted $2.01 \cdot 10^4$ nodes (these are the not bounded ones) and expanded $1.33 \cdot 10^4$ nodes during a tour construction in average. The total time of the algorithm was about $2.6 \cdot 10^3$ minutes. The heuristics was calculated in about $26.3$ minutes in total.
The new branch and bound procedure (with the B&B heuristic) expands about $1.85 \cdot 10^4$ nodes, inserted $1.18 \cdot 10^4$ nodes and expanded $7.22 \cdot 10^4$ nodes during an tour construction in average. This means that the new heuristic reduces the number of node expandations by $55\%$. However, each node creation needs more time. The heuristics was calculated in $927$ minutes in total ($35$ times more than the original). Fortunately the total time of the algorithm was about $3.67 \cdot 10^3$ minutes, which is just $40\%$ more than the original.

### 4.2 Costs

The following numbers are not exactly the profits of the tours, because some costs are calculated in other parts of the software (such as the task assignment). Some costs are also modified for the algorithm (only without changing the optimality of the plans), for example the fuel costs and the wages of the drivers.
The original algorithm results $1.03 \cdot 10^9$ objective function value in total, and $713$ Euros of costs in average. The new algorithm results $9.69 \cdot 10^8$ objective function value in total, and $686$ Euros of costs in average. This means that we reach about $4\%$ better results with the new algorithm.
These results are promising. Moreover the new solution will be even better for longer tours. We are not capable to make statistics for more than 10 routing tasks in one plan yet. However, with the new algorithm it could be profitable to add more computational capacity, and that way plan more routing tasks in one run.

### Acknowledgement

This work is connected to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project, supported by the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002) Research has also been partially supported by grant # 124171 of the Hungarian National Research, Development and Innovation Office.

## References

[1] M. W. Carter, C. C. Price (2000) *Operations research: a practical introduction*, Crc Press. https://doi.org/10.1201/9781315274188

[2] C. S. Chung, J. Flynn, Ö. Kirca, (2006) A branch and bound algorithm to minimize the total tardiness for m-machine permutation flowshop problems. *European Journal of Operational Research*, 174(1), 1–10. https://doi.org/10.1016/j.ejor.2004.12.023

[3] R. Companys, M. Mateo (2007) Different behaviour of a double branch-and-bound algorithm on Fm| prmu| Cmax and Fm| block| Cmax problems. *Computers & Operations Research*, 34(4), 938–953. https://doi.org/10.1016/j.cor.2005.05.018

[4] Cs. Gy. Csehi, M. Farkas (2016) Truck routing and scheduling, Central *European Journal of Operations Research*, 1–17.
https://doi.org/10.1007/s10100-016-0453-8

[5] Y. Cui, Y. Yang, X. Cheng, P. Song (2008) A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. *Computers & Operations Research*, 35(4), 1281–1291.
https://doi.org/10.1016/j.cor.2006.08.011

[6] R. W. Floyd (1962) Algorithm 97: Shortest path, *Communications of the ACM* 5(6):345
https://doi.org/10.1145/367766.368168

[7] A. Hartwig (1985) Recursive branch and bound. *Optimization*, 16(2), 219–228.
https://doi.org/10.1080/02331938508843011

[8] A. Hartwig, F. Daske, S. Kobe (1984) A recursive branch-and-bound algorithm for the exact ground state of Ising spin-glass models. *Computer Physics Communications*, 32(2), 133–138.
https://doi.org/10.1016/0010-4655(84)90066-3

[9] R. Matsuzaki, A. Todoroki (2007) Stacking-sequence optimization using fractal branch-and-bound method for unsymmetrical laminates. *Composite Structures*, 78(4), 537–550.
https://doi.org/10.1016/j.compstruct.2005.11.015

[10] M. Vanhoucke (2002) Optimal due date assignment in project scheduling. *Working Papers of Faculty of Economics and Business Administration, Ghent University, Belgium*
02/159, Ghent University, Faculty of Economics and Business Administration.
https://ideas.repec.org/p/rug/rugwps/02-159.html

[11] M. Vanhoucke (2006) Scheduling an R&D project with quality-dependent time slots. *International Conference on Computational Science and Its Applications* (pp. 621–630). Springer Berlin Heidelberg.
https://doi.org/10.1007/11751595_66

[12] S. Warshall (1962) A theorem on Boolean matrices, *Journal of the ACM* 9(1):11–12
https://doi.org/10.1145/321105.321107