

VRP によるテストスケジュールの自動作成

Automated Test Scheduling Using Vehicle Routing Problem Methodology

蟻 坂 篤 史¹
Atsushi ARISAKA

宮 崎 武 司¹
Takeshi MIYAZAKI

福 井 一 輝²
Kazuki FUKUI

要 旨

新型車開発の期間短縮と開発費低減のため、テスト車台数を最小化し最短のテスト日程を算出する実車テストスケジュールの自動作成手法を考案した。衝突テストのスケジューリングに数理最適化手法を用い、機能テストのスケジューリングには、特に数理最適化手法の中の配送計画問題の解法を用いることで、考慮すべき制約条件を自動計算に組み込むことができた。表計算ソフトと簡易アプリケーションを作成し手法の有効性と有用性を検証した。その結果、従来エキスパートが90時間を要していた作業が、6時間でできるようになった。

ABSTRACT

In order to shorten the development period and help reduce development costs for new vehicle models, a method for automated scheduling of actual vehicle tests was devised that helps to minimize the number of test vehicles and calculate the shortest test schedule. Use of mathematical optimization methodology for collision test scheduling and use of the vehicle routing problem solution in particular in mathematical optimization methodology for function test scheduling enabled incorporation of the constraints into the automated calculations to be considered. Simple applications were created and verified using the applications with a spreadsheet software. As a result, a scheduling task that previously took an expert 90 hours could be accomplished in 6 hours.

1. ま え が き

近年、CAE技術、性能設計技術などの進化により、多くの性能が机上で検討できるようになっている⁽¹⁾⁻⁽³⁾。また、自動車各社は性能向上に加え開発費を抑制するため、開発期間短縮とテスト車台数の削減も進めている。

しかし、すべての性能事象を予測することは容易でなく、法規により実車による確認が義務付けられている項目があるため、量産前には実車による確認テストが実施されている。

この確認テストの数は、法規の強化と、仕向地、エンジン、トランスミッション、制御などの仕様、装備の多様化により、さらに増加傾向となっている。

これまで衝突試験設備を用いた確認テスト（以下、衝突テスト）のためのテスト車と衝突試験設備を用いずに実施する性能確認テスト（以下、機能テスト）のためのテスト車は別々に取り扱われていたが、最近ではテスト車台数を削減するため、機能テストを実施した車両を衝突テストに使用することが要望されている。

さらに、衝突テストでは同時に開発する機種増加に伴い衝突試験設備の調整が課題となり、機能テストでは様々な確認項目の実施時期や順序の制約によりスケジュールの調整が課題となる。

そのため、事前にテストスケジュールを作成し、すべての機能テストと衝突テストを試験設備とテスト車に割り当て、定められた期間内（以後、テスト期限）に完了できるかどうかを検討している。

しかし、実際の車両開発では、比較的バリエーションの少ないフルモデルチェンジでも約50台のテスト車で約1000の確認テストが実施され、かつ、一部の機能テストはテスト担当者から個々に要望される場合が多い。そのため、テストスケジュールの作成には経験ある人でも2週間から1ヵ月を要しており、テストスケジュール作成が新型車開発における課題となっている。

これまでに、テスト車台数、そして、確認テスト数と各テストに必要な日数と順序を制約として与えると最短日程の衝突テストスケジュールを自動的に作成する手法が報告されている^{(4), (5)}。

¹ 本田技研工業株式会社

E-mail: atsushi_arisaka@jp.honda

² 株式会社フレクト

この手法は、テストスケジュールの作成に Operations Research (OR) における数理最適化の手法を用い、限られた数の確認テストのスケジュールを自動的に作成することができる。Fig. 1 に、7 台のテスト車で 4 項目の衝突テストと機能テストを実施するスケジュールの例を示す。

この手法では、テスト A と D を最初に実施するなどの簡単な条件を考慮することができるが、衝突テストの後に機能テストを実施することはできないため、機能テストは別のテスト車でスケジュールを計画しなければならない。

また、機能テストの中には、最初に実施することが必要なテスト、複数のテストの間で実施の順序が決められているテスト、実施時期を集約することによりテスト期間が短縮できるテストなど複雑な条件を考慮しなければならない。

そのため、このままでは実際の開発での衝突テストに機能テストを加えた全体のテストスケジュールを自動的に作成することはできない。

そこで、本研究では、衝突設備の数、テスト車台数を計算のための条件とし、衝突テストと機能テストそれぞれの数と各テストに必要な日数、そして、テストの順番を与えることにより、すべてのテストが考慮された最短日程のテストスケジュールを自動的に作成する手法について検討した。

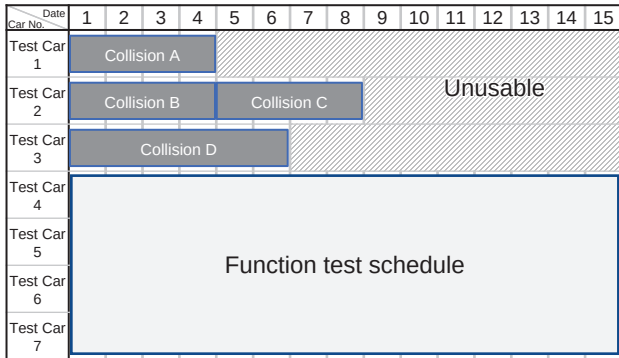


Fig. 1 Example of collision and function test schedule

2. 衝突テストのスケジューリング

2.1. テスト重複の防止

一台のテスト車では複数のテストを同時に実施することはできない。そこで、各テスト車において実施されるテストの期間の重なりを禁止するための制約条件は式 (1) となる。

$$\text{Non-overlap}(\{I(s_t, P_t, x_{t,v})\}_{t \in \mathcal{T}}), \forall v \in \mathcal{V} \quad (1)$$

ここで $I(s, P, x)$ は、テスト開始日時 s からテスト日数 P の期間を表す interval variable であり、 t はテスト、 s_t はテ

スト t の開始日時を表す変数、 P_t はテスト t のテスト日数を表す定数、 $x_{t,v}$ はテスト t がテスト車 v で実施されると 1、されないと 0 となる二値変数、 $\mathcal{T}$ はテストの集合、 $\mathcal{V}$ テスト車の集合である。Non-overlap($\cdot$) は、global constraint⁽⁶⁾ と呼ばれる特殊な制約条件の一つで、与えられた interval variable の重複を許容しない制約条件である。

この制約条件により、複数のテストを同時に一台のテスト車に割り当ててを防ぐことができる。

2.2. テスト開始時期の設定

テストにはテスト車製作直後に実施できる場合と、必要な部品がそろわないと実施できない場合がある。

テスト車 v が利用可能になる日時を納車日 Q_v とすると、 Q_v は、次式を満足する必要がある。

$$x_{t,v} = 1 \Rightarrow s_t \geq Q_v, \quad \forall t \in \mathcal{T}, \quad \forall v \in \mathcal{V} \quad (2)$$

(2) 式の $\Rightarrow$ は、論理包含を表しており、左辺で示す、テスト t がテスト車 v に割り当てられたときは、右辺の制約条件が満たされること、すなわち、テスト t の開始日時をテスト車 v の納車日時以降にすることを意味する。

部品がそろわないと実施できない場合の制約条件は次式となる。

$$s_t \geq R_t, \quad \forall t \in \mathcal{T} \quad (3)$$

ここで、 R_t はテスト t が開始可能となる日時を表す定数であり、設備が使えるようになる日時やテストに必要な部品が納品される日時等を設定する。

これらの条件設定により、テスト開始可能時期を設定することができる。

2.3. テスト車のダメージの考慮

テスト実施以前の別の衝突テストにより、テスト車にダメージが入る場合、実施できないテストがある。

まず、決定変数間の整合性を取るために次式の制約条件を追加する。

$$x_{t,v} = 1 \Rightarrow w_t = v, \quad \forall t \in \mathcal{T}, \quad \forall v \in \mathcal{V} \quad (4)$$

ここで、 w_t はテスト t に割り当てられるテスト車を表す変数である。

式 (4) は、テスト t がテスト車 v に割り当てられたときは、そのテスト車は w_t に設定されることを示す。さらに、式 (5) の制約条件を設定することで、同一テスト車でテスト t_1 と t_2 をおこなう場合、ダメージが入る t_1 の以前にしか、 t_2 が実施できなくなる。

$$w_{t_1} = w_{t_2} \Rightarrow s_{t_2} + P_{t_2} \leq s_{t_1} \quad (5)$$

以上より、テスト前のテスト車のダメージ状態を制約条件として考慮できるようになる。

2.4. テスト車台数の適正化

テスト車台数を最小とするためのペナルティの総和を表す目的関数は式 (6) となる。

$$\min \sum_{v \in V} C_v^V u_v \quad (6)$$

$$x_{t,v} \leq u_v, \quad \forall t \in T \quad (7)$$

ここで、 C_v^V はテスト車 v を衝突テストに使用した際に課されるペナルティ定数を表し、 u_v はテスト車 v が衝突テストに使用されたかどうかを示す二値変数である。 u_v の値は衝突テストに使用される場合は 1 に、されない場合 0 となる。式 (7) の制約で、衝突テストがテスト車 v に割り当てられると $u_v = 1$ となり、式 (6) のペナルティがおおきくなる。

上記の目的関数と制約により、テスト車を最小限にすることができる。

2.5. テスト完了期限の考慮

テスト車台数を減らすためには、一台のテスト車で、できる限り多くのテストをテスト期限までに実施することが望まれる。そのために、テスト期限を考慮する目的関数の式 (8) を設定する。

$$\min \sum_{t \in T} C_t^T o_t \quad (8)$$

$$s_t + P_t \leq D_t + o_t, \quad t \in T \quad (9)$$

ここで、 C_t^T はテスト t のテスト期限からの遅れに対して課されるペナルティを表す定数、 D_t はテスト項目 t のテスト期限を表す定数、 o_t はテスト t のテスト期限からの遅れを表す変数である。

式 (9) の制約により各テストの遅れが o_t として表される。式 (8) で、遅れの合計が最小化されるため、すべてのテストをテスト期限内に割り当てることができる。

2.6. 各衝突テストの順序

一方で、衝突テストには、乗員保護、安全装備、軽衝突などのテストがあり、実施順序が決まっている。その順序に従ってテストを実施する際、Fig. 2 の U のように一定期間以上の間隔を設ける必要もある。

そこで n 番目のテスト t_n と $n+1$ 番目のテスト t_{n+1} の間に間隔を設けるために式 (10) の制約条件を追加した。

$$s_{t_n} + U_{t_n, t_{n+1}} \leq s_{t_{n+1}} \quad (10)$$

ここで、 $U_{t_n, t_{n+1}}$ は Fig. 2 に示したような n 番目のテスト t_n と $n+1$ 番目のテスト t_{n+1} の間に設けるオフセットの長さを表す定数である。

この条件により、必要な間隔をあけた上で、各衝突テストの順序が決められるようになった。

2.7. 衝突試験設備の数の考慮

衝突試験設備の数には限りがあり、同一のリソースを使用する複数のテストに上限を設ける必要がある。

そこで、使用する設備の総数を制約条件として考慮すると式 (11) となる。

$$\text{Cumulative}(\{I(s_v, P_v, 1)\}_{t \in T, g}, L_g), \quad \forall g \in G \quad (11)$$

$\text{Cumulative}(\{I_t\}_t, L)$ は、 $\{I_t\}_t$ で与えられた interval variable の重複を L 回以下に抑える制約条件である。

ここで、 G はリソースの集合、 $T_g \subset T$ はリソース $g \in G$ を使用するテスト項目の集合である。

一般的には式 (11) にて表現できるが、衝突テストの具体的な日程を考えたとき Fig. 3 に示すように、準備、衝突テスト、後処理の 3 工程から構成されることが多い。そのため、テスト期間に対して衝突テストそのものは実質 1 日となる。また、1 日に 2 回まで実施可能とすると、式 (11) の制約条件は式 (12) のようになる。

$$\text{Cumulative}(\{I(s_t + P_t^{\text{Prep}}, P_t^{\text{Hit}}, 1)\}_{t \in T}, 2) \quad (12)$$

ここで、 P_t^{Prep} は Fig. 3 の Preparation であり、テスト t の準備期間を表す。 P_t^{Hit} は Fig. 3 の Hit である衝突テストの期間を表す。

この条件の追加により試験設備の数を考慮できるようになった。

2.8. 機能テストのための日程の確保

衝突テストに使用するテスト車を機能テストに使用する

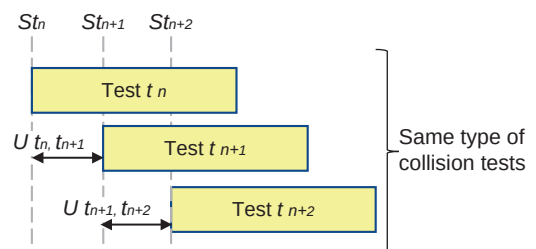


Fig. 2 Sequence of collision tests and start-time offsets

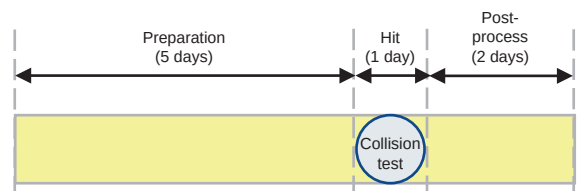


Fig. 3 Schedule for front collision tests

る場合は、衝突テスト後には機能テストが実施できないため、衝突テストの前に実施する必要がある。

そのためには Fig. 4 における r_t を小さくし、できる限りテストの完了時刻をテスト期限に寄せて、衝突テスト前の機能テスト実施可能な日数を長くすることが必要となる。そこで、目的関数、制約条件式 (13) と式 (14) を設定した。

$$\min \sum_{t \in T} C_t^R r_t \quad (13)$$

$$D_t - s_t - P_t \leq r_t, \quad \forall t \in T \quad (14)$$

ここで、 D_t はテスト期限、 r_t はテスト t による減少するテスト可能日数を表す変数、 P_t はテスト日数である。

そして、減少するテスト可能日数に対するペナルティを表す定数を C_t^R とする。

この目的関数と制約条件により衝突テスト前の機能テスト実施可能な日数をできるだけ多くするように衝突テストを割り当てることができた。

以上の制約条件と目的関数を OR の中で考慮することにより、Fig. 5 に示すような機能テストのための日程を確保した衝突テストのスケジュールが自動作成できるようになった。

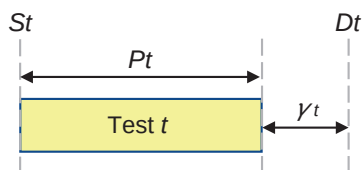


Fig. 4 Reduction of vehicle-days due to collision tests

Date	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Car No.															
Test Car 1															
Test Car 2															
Test Car 3															
Test Car 4															
Test Car 5															
Test Car 6															
Test Car 7															

Fig. 5 Example of collision test schedule with days available for function tests

3. 機能テストのスケジューリング

3.1. アプローチ

機能確認のためのテストを順番に各テスト車に割り当

ていく作業は、Fig. 6 上段のようなあるテスト車がテスト実施後に、どのテストを実施するかをの遷移を表すネットワークに類似していると考えられた。

例えば、Start と End はテスト車の使用開始と終了を表すノードであり、その他のノードがテスト項目を表すと解釈すると、このネットワーク上の Start から End に至るまでのノードを通る順番は、Fig. 6 下段のようなテスト計画で表現できる。

その際、遷移に対するペナルティの大きさを設定する。例えば、Fig. 7 で Test A から B への遷移のように同一部門間の遷移場合はテスト車の受け渡し作業が無いため、ペナルティを小さくし、Test A から C, D へのように部門が異なる場合はペナルティを大きくする。

このアプローチにより、テスト車に各テストが効率的に割り振られるテストスケジュールを見つけることが、配送車がどのようなルートで配送を完了させるかを、ペナルティの合計の大きさを判断する数理最適化手法の一つである配送計画問題 (Vehicle Routing Problem: VRP) ⁽⁷⁾ と類似していることがわかった。そのため、新たな取り組みとし

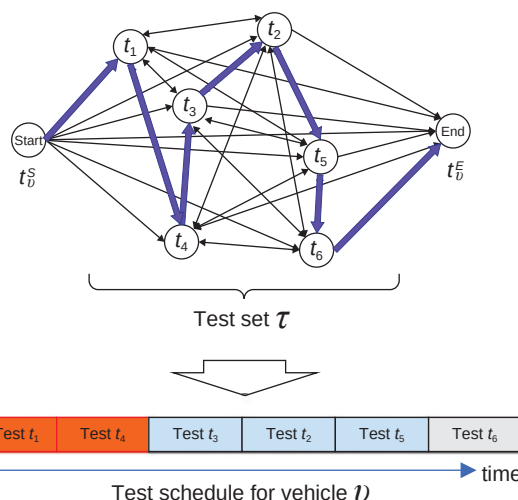


Fig. 6 Correspondence between test network and test schedule

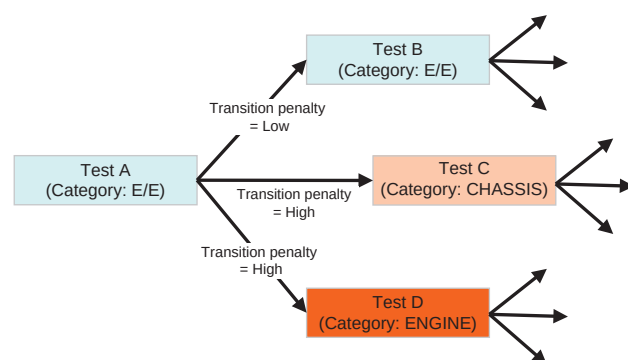


Fig. 7 Transition penalties

て VRP の手法を機能テストのスケジューリングに取り入れることとした。

3.2. 重複テストの防止

衝突テスト同様に一台のテスト車で複数のテストを同時に実施することはできない。

式 (15) (16) はテストネットワーク上で、始めのテストから最後のテストまでの間、かならず一つのテストから他の一つのテストに遷移することを表す制約条件である。

$$\sum_{a \in \mathcal{A}_t^O} y_a = 1, \quad \forall t \in \mathcal{T} \cap \{t_v^S\}_{v \in \mathcal{V}} \quad (15)$$

$$\sum_{a \in \mathcal{A}_t^I} y_a = 1, \quad \forall t \in \mathcal{T} \cap \{t_v^E\}_{v \in \mathcal{V}} \quad (16)$$

$\mathcal{A}_t^O$ はテスト t のノードから出る有向辺の集合、 $\mathcal{A}_t^I$ はテスト t のノードに入る有向辺の集合、 y_a は有向辺 a を通るなら 1、通らない場合 0 となる二値変数、 t_v^S は車両 v の開始ノード、 t_v^E は車両 v の終了ノードを表す。

この条件により、テストの重複を防ぐことができるようになった。

3.3. テスト開始時期の設定

テストが実施される日時の経過は、式 (17) から (20) で表される。これらは、初めのテストから最後のテストまでの時間変化の整合性を表す制約条件である。

$$y_{(t_v^S, t)} = 1 \Rightarrow s_t \geq Q_v, \quad \forall t \in \mathcal{T}, \quad \forall v \in \mathcal{V} \quad (17)$$

$$y_{(t_1, t_2)} = 1 \Rightarrow s_{t_1} + P_{t_1} \leq s_{t_2}, \quad \forall t_1, t_2 \in \mathcal{T} \quad (18)$$

$$y_{(t, t_v^E)} = 1 \Rightarrow s_t + P_t = e_v, \quad \forall t \in \mathcal{T}, \quad \forall v \in \mathcal{V} \quad (19)$$

$$y_{(t_v^S, t_v^E)} = 1 \Rightarrow e_v = Q_v, \quad \forall v \in \mathcal{V} \quad (20)$$

ここで Q_v は納車日時、 e_v は車両 v の利用終了日時を表す変数である。

この中で式 (17) が納車後にしかテストができない制約条件である。

上記制約条件によりテストの開始を定義することができた。

3.4. ダメージの考慮

衝突テスト実施後にテストができない場合、あるいは、ダメージが生じる機能テストの後にも実施ができない場合がある。まず、式 (21) から (23) の制約条件にて、テストネットワーク上のテスト車のフローとテスト t に対するテスト車の割り当てを表す変数 w_t との整合性を定義する。

$$y_{(t_v^S, t)} = 1 \Rightarrow w_t = v, \quad \forall t \in \mathcal{T}, \quad \forall v \in \mathcal{V} \quad (21)$$

$$y_{(t, t_v^E)} = 1 \Rightarrow w_t = v, \quad \forall t \in \mathcal{T}, \quad \forall v \in \mathcal{V} \quad (22)$$

$$y_{(t_1, t_2)} = 1 \Rightarrow w_{t_1} = w_{t_2}, \quad \forall t_1, t_2 \in \mathcal{T} \quad (23)$$

上記の式、および前述の式 (5) によって、テスト実施におけるテスト車へのダメージを考慮することができた。

以上の VRP の基本的なテスト車のテスト間の遷移、日時の整合性を定義する制約条件⁽⁸⁾によって、重複テストの防止、テストの開始日時、ダメージを受けたテスト車の考慮を取り入れた上で、すべてのテスト車にすべてのテストが割り当てられるようになった。

3.5. 優先度の高いテストの取り扱い

機能テストには、空力テストのように他のテスト条件に影響するもの、エンジン性能の初期確認のように、他のテスト自体に影響するものなど、先に実施しなければならないものがある。

このような優先度の高いテストをできるだけ先に実施するために、目的関数式 (24) と制約条件式 (25) を設定した。ここで $\mathcal{T}^P$ は優先度の高いテストの集合、 C_t^P は優先度の高いテスト t の開始日時の遅れに対して課されるペナルティを表す定数、 B_t は優先度の高いテスト t を開始可能となる一番早い日時を表す定数、 o_t^P は優先度の高いテスト t の開始日時 s_t と一番早い開始可能日時 B_t との日時差を表す変数である。

$$\min \sum_{t \in \mathcal{T}^P} C_t^P o_t^P \quad (24)$$

$$s_t - B_t \leq o_t^P \quad (25)$$

これらにより、優先度の高いテストをできるかぎり先に割り当てることができた。

3.6. 実施順序が決まっているテストの取り扱い

テストによっては Fig. 8 のテスト J と K のように、実施

Date	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Test Car 1						Collision A			Unusable						
Test Car 2	High priority tests							Collision B			Collision C				
Test Car 3	Function E							Collision D							
Test Car 4	Tests with decided order														
Test Car 5						Function J				Function K Must conduct after J					
Test Car 6	Function L T section			Function M T section			Function N T section			Function O T section					
Test Car 7	Function P		Same category tests												

Fig. 8 Arrangement of function tests

順序が決まっているものがある。例えば、サスペンションのセッティングをおこなった後に、走行性能のテストを実施するような場合がある。このような場合は、二つのテスト t_1, t_2 の実施順序を、式 (26) の制約条件にて定めることができた。

$$s_{t_1} + P_{t_1} + M_{t_1, t_2} \leq s_{t_2} \quad (26)$$

ここで、 M_{t_1, t_2} はテスト t_1 から t_2 を実施するまでの期間を表す定数である。

さらに、テスト t_1, t_2 を同じテスト車で実施する必要がある場合には、制約条件 式 (27) を追加する。

$$w_{t_1} = w_{t_2} \quad (27)$$

式 (26), (27) によって、同一テスト車で連続テストを割り当てることができた。

3.7. 同一部門の連続テストによる効率化

前述の通り、テスト車を使用する部門が切り替わると部門間でのテスト車の引き渡しのための業務負荷が発生する。よって同じ部門のテストはできるだけ同じテスト車を連続して割り当てるようにする。Fig. 8 の同一部門のテストがこれにあたる。

このために、テスト間の遷移の選択に対するペナルティを適切に設定し、できるだけ部門の切り替わりが少なくなるように、式 (28) の目的関数を設定した。

$$\min \sum_{a \in \mathcal{A}} C_a^A y_a \quad (28)$$

ここで、 $a = (t, t') \in \mathcal{A}$ はテスト t からテスト t' への遷移を表す有向辺の集合、 C_a^A は有向辺 a で表されるテストの遷移に伴い発生するペナルティを表す定数である。

上記目的関数の設定により、部門間の切り替えの数を減らすことができた。

3.8. テスト期限内完了に近づける対応

全てのテストはテスト期限内に完了するようにスケジュールリングされることが望まれる。しかし、テストの数とテスト間での条件が多いため、すべてのテストが収まりきらず、Fig. 9 のように当初計画したテスト期限からはみ出す場合がある。

各テストのテスト期限からの遅れの合計を最小化するために、衝突テストでの式 (8) の完了期限を考慮する目的関数に対して、さらに、全体で一番遅い終了日時を最小化する条件を追加した。

$$\min \max\{e_v\}_{v \in V} \quad (29)$$

これより、すべてのテストを最も早く完了させる結果が得られるようになった。

機能テストスケジュールリングにおいて、VRP の手法を取り入れ、VRP の基本的なテスト車のテスト間の遷移、日時の整合性を定義する制約条件を設定し、さらに機能テスト特有の制約条件と目的関数を追加することにより、機能テストの遷移の条件が満足されている機能テストスケジュールの自動化ができた。

Date	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Test Car No. 1	Function A		Function B		Collision A				Unusable						
Test Car 2	Function C			Function D			Collision B				Collision C				
Test Car 3	Function E	Function F		Function G			Collision D								
Test Car 4	Function H														
Test Car 5	Function I				Function J				Function K						
Test Car 6	Function L			Function M				Function N				Function O			
Test Car 7	Function P		Function Q					Function R					Function S		

Fig. 9 Example of schedule exceeding deadline

4. 検証

実際に計算を実施する場合には、計算に必要な条件を与え、計算結果を確認するための入出力機能と、OR と VRP の数理最適化の計算を実施するためのアプリケーションの作成が必要となる。

そこで、入出力画面に表計算ソフトを用い、テスト車の情報、テストの情報、テストスケジュールに関わる条件の情報を記載できるようにした。

そして、OR と VRP の数理最適化の計算のためのアプリケーションをそれぞれプログラム言語で作成することとした。プログラム言語には一般的に普及している Python を用いた。計算の実行には計算機 (Xeon Platinum 8000 クロック周波数 3.6 GHz メモリ 8 GB) を用いた。

計算結果の一例を Fig. 10 に示す。

これは、比較的バリエーションの少ないフルモデルチェンジの例であるが、それでも 45 台のテスト車と 931 個のテストをスケジュールリングしたものである。

過去にエキスパートが手作業で作成した際には 90 時間以上を要していたが、前述の仕様の計算機で計算したところ、約 6 時間でテストスケジュールが作成できた。

計算時間はテスト車台数、テスト数の影響で増減することが考えられるが、Fig. 11 に示すように、テストスケジュール作成を OR および VRP の数理最適化手法によって自動化することにより、計算時間がエキスパートの手作業に対し 90% 以上低減できることが確認できた。



Fig. 10 Calculation result

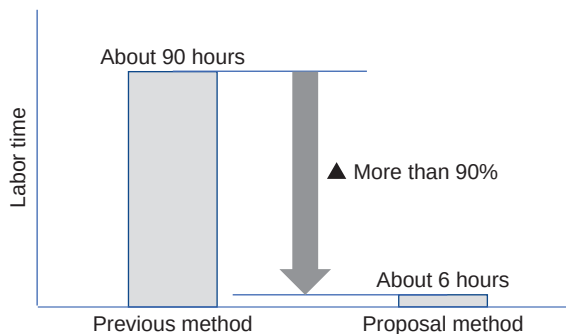


Fig. 11 Comparison between previous scheduling time and proposal scheduling time

5. まとめ

本研究では、衝突設備の数、テスト車台数を計算のための条件とし、衝突テストと機能テストそれぞれの数と各テストに必要な日数、そして、テストの順番を与えれば、最短日程のテストスケジュールを自動的に作成する手法について検討した。結果を以下にまとめる。

- (1) OR と特に VRP の数理最適化手法を取り入れ、最短日程のテストスケジュールを自動的に作成する手法を確立した。
- (2) OR 手法にて衝突テストの制約条件と目的関数を定義し、機能テストのための日程を確保した衝突テストスケジュールを自動作成できるようになった。
- (3) VRP の基本的なテスト車のテスト間の遷移、日時の整合性を定義する制約条件と、機能テスト間の条件を表す制約条件、目的関数を定義し、機能テストスケジュールを自動作成できるようになった。

- (4) 自動作成によるテストスケジュールの検証のために、衝突テスト、機能テストの全てのテスト条件を考慮したテストスケジュール自動作成アプリケーションを作成した。
- (5) 既存機種のデータ情報を用いてテストスケジュールの自動作成をおこない、作成時間が 90% 以上短縮されたことが確認できた。

参考文献

- (1) 豊島貴行, 松澤俊明, 伊藤憲悟, 遠藤喜章, 高野広喜, 牛村智也: 量産車の MBD におけるシャシ性能設計技術の構築, Honda R&D Technical Review, Vol. 29, No. 1, p. 49-58
- (2) 藤本茂希, 東上圭司, 楠祥治, 齊藤徹: DSM とシミュレーションを用いた開発プロセスの提案, Honda R&D Technical Review, Vol. 29, No. 1, p. 99-108
- (3) 安原重人, 吉本毅, 岡田英之: 四輪開発業務における生産性向上のためのエキスパートの知識 経験 思考の構造化と活用, Honda R&D Technical Review, Vol. 32, No. 2, p. 89-96
- (4) Reich, D., Shi, Y., Epelman, M., Cohn, A., Barnes, E., Arthurs, K., Klampfl, E.: Scheduling Crash Tests at Ford Motor Company, Interfaces, Vol. 46, No. 5, p. 365-462, (2016), doi: 10.1287/inte.2016.0855
- (5) Shi, Y., Reich, D., Epelman, M., Klampfl, E., Cohn, A.: An analytical approach to prototype vehicle test scheduling, Omega, Vol. 67, p. 168-176, (2017), doi: 10.1016/j.omega.2016.05.003
- (6) Beldiceanu, N., Carlsson, M., Rampon, J-X.: Global

Constraint Catalog, HAL, 1406p., (2005), <https://hal.science/hal-00485396>

- (7) Braekers, K., Ramaekers, K., Nieuwenhuyse, I. V.: The vehicle routing problem: State of the art classification and review, *Computers & Industrial Engineering*, Vol. 99, p. 300-313, (2016), doi: 10.1016/j.cie.2015.12.007
- (8) El-Sherbeny, N. A.: Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods, *Journal of King Saud University - Science*, Vol. 22, Issue 3, p. 123-131, (2010), doi: 10.1016/j.jksus.2010.03.002

■ 著 者 ■



蟻坂 篤史



宮崎 武司



福井 一輝