

BenchQC - Anwendungsfälle der Optimierung

OptWare GmbH

24. April 2024

Zusammenfassung

Das Forschungsprojekt BenchQC, Anwendungsgetriebenes Benchmarking von Quantencomputern, befasst sich mit echten Anwendungsfällen der Industrie aus Machine Learning, Simulation und Optimierung. In diesem Dokument sollen die folgenden industriellen Anwendungsfelder der Optimierung betrachtet werden:

- Fließbandabstimmung
- Reihenfolgeplanung
- Testfahrzeugkonfiguration
- ...

Die einzelnen Anwendungsfälle werden zunächst definiert, ihre klassischen Algorithmen erarbeitet und anschließend für die Anwendung auf Quantencomputern modifiziert. Abschließend werden die klassischen und QC Lösungen mithilfe eines Benchmarking Frameworks (QUARK) verglichen.

Inhaltsverzeichnis

1	Anwendungsfälle	2
1.1	Fließbandfertigung	3
1.1.1	Anwendungsfall Fließbandabstimmung	3
1.1.2	Anwendungsfall Reihenfolgeplanung (Sequencing)	6
1.2	Anwendungsfall Testfahrzeugkonfiguration	10
1.2.1	Transformation Testinkompatibilitäten	12
1.2.2	Transformation zu Bin Packing mit Konflikten	13
2	Mathematische Grundprobleme	15
2.1	Bin Packing	15
2.1.1	Assembly Line Balancing Problem	17
2.1.2	Bin Packing mit Konflikten	19
3	Problem mappings	20
3.1	Bin packing with conflicts - QUBO model	20
4	Metriken	22
4.1	Berechnungszeit	22
4.2	Lösungsqualität	22
4.3	Kombinationen der Metriken	23
4.3.1	Time-to-solution	23
4.3.2	Q-Pack	24
4.4	Energie-verbrauch / -kosten	24
4.5	Hardware-spezifische Metriken	24
4.5.1	Quantum Volume	24
4.5.2	Gate Error Rates	25
5	Versionshistorie	26
	Liste der noch zu erledigenden Punkte	27

1 Anwendungsfälle

Im Folgenden sollen zunächst die industriellen Anwendungsfälle vorgestellt werden.

1.1 Fließbandfertigung

Moderne Produktion, wie zum Beispiel die Automobilproduktion, erfolgt oft mithilfe der Fließbandfertigung. Bei der Fließbandfertigung durchläuft das zu fertigende Produkt auf dem Weg zur Fertigstellung zahlreiche Arbeitsstationen, in denen die einzelnen Arbeitsschritte zur Fertigstellung des Produktes ausgeführt werden.

Die Beplanung einer Fertigungsline erfolgt dabei in zwei Schritten: Die Fließbandabstimmung (cf. Abschnitt 1.1.1) und die Reihenfolgeplanung (cf. Abschnitt 1.1.2).

Die Fließbandabstimmung (*Assembly Line Balancing*) beschäftigt sich damit, Arbeitsschritte nach ihrem mittleren Zeitbedarf auf Arbeitsstationen zu verteilen. In der Reihenfolgeplanung (*Sequencing*) werden konkrete Folgen von Aufträgen gebildet, die in Form einer operativen Feinplanung zu große Abweichung von den in der Fließbandabstimmung verwendeten Mittelwerten auf den einzelnen Stationen verhindern sollen. Die Fließbandabstimmung liefert somit eine taktische Grobplanung, auf der die operative Reihenfolge(fein)planung basiert.

Der mittlere Zeitbedarf der Arbeitsschritte, die als Ausgangsbasis der Fließbandabstimmung dient, ergibt sich aus dem tatsächlichen Zeitbedarf für einen Arbeitsschritt multipliziert mit der erwarteten Häufigkeit des Auftretens dieses Schrittes. Dauert z.B. die Montage einer Anhängerkupplung 100 Sekunden, und ist eine Anhängerkupplung nur bei jedem zehnten Fahrzeug zu erwarten, beträgt der mittlere Zeitbedarf 10 Sekunden.

1.1.1 Anwendungsfall Fließbandabstimmung

Gegeben ist bei dem Problem der Fließbandabstimmung eine Reihe von Arbeitsschritten, die Arbeitsstationen zugeordnet werden sollen. Dabei soll eine möglichst geringe Anzahl und eine im Mittel gleichmäßige Auslastung der Arbeitsstationen angestrebt werden. Pro Arbeitsstation steht maximal ein Takt zur Verfügung. Zum Taktschlag sollen die Arbeitsaufträge auf der nächsten Station bearbeitet werden. Somit darf die summierte durchschnittliche Arbeitszeit der Arbeitsschritte die Taktzeit nicht überschreiten.

Die Arbeitsschritte müssen teilweise in einer vorgegebenen Reihenfolge abgearbeitet werden. Abbildung 1 zeigt im oberen Teil einen beispielhaften

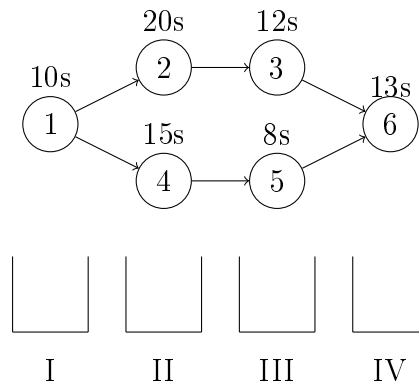


Abbildung 1: Beispiel für Fließbandabstimmung: Die einem Reihenfolgegraph unterliegenden Arbeitsschritte (1-6) sollen nach Zeitbedarf gleichmäßig auf Arbeitsstationen (I-IV) verteilt werden. Dabei darf die Arbeitslast pro Station 25 Sekunden nicht überschreiten.

Reihenfolgegraphen mit den Tätigkeiten (1-6) zu denen auch jeweils eine mittlere Dauer (z.B. 10 Sekunden bei Tätigkeit 1) angegeben ist. Tätigkeit 2 hängt von Tätigkeit 1 ab und kann deshalb erst durchgeführt werden, wenn Tätigkeit 1 bereits abgeschlossen ist. Gleiches gilt für Tätigkeit 3, sie hängt wiederum von Tätigkeit 2 ab. Tätigkeit 4 im Beispiel kann parallel zu Tätigkeit 2 erfolgen, da sie nur von Tätigkeit 1 abhängig ist und nicht etwa von Tätigkeit 2 oder 3.

In Abb. 1 sind im unteren Teil Arbeitsstationen (I-IV) als offene Behälter angedeutet. Den Arbeitsstationen sollen nun, unter Achtung des Reihenfolgegraphen, Tätigkeiten zugeteilt werden. Die Taktzeit beträgt 25 Sekunden. Die Zuordnung in Abb. 2 wäre denkbar, während die Zuordnung in Abb. 3 nicht möglich ist, da sie dem Reihenfolgegraphen widerspricht und Station III die Taktzeit überschreitet.

Transformation zu Bin Packing

Das Fließbandabstimmungsproblem lässt sich klassisch als Assembly Line Balancing Problem aus Abschnitt 2.1.1 aufstellen und lösen. Die Arbeitsschritte sind dabei die Aufgaben und die Arbeitsstationen die Stationen des klassischen Problems. Die Taktzeit wird durch die maximale Zeit pro Station dargestellt.

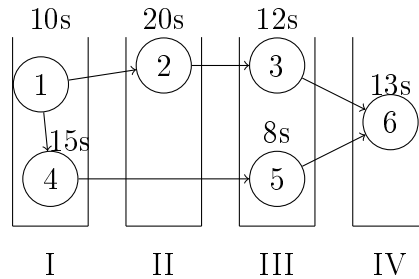


Abbildung 2: Mögliche Lösung für das Ausgangsproblem aus Abb. 1

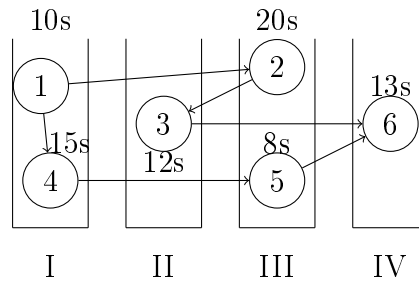


Abbildung 3: Unzulässige Lösung für das Ausgangsproblem aus Abb. 1

Zusammengefasst, lässt sich die Fließbandabstimmung wie folgt auf die Notation aus Abschnitt 2.1.1 mappen:

Symbol	Bedeutung	Definitionsbereich
Mengen		
T	Menge der Arbeitsschritte	\mathbb{N}
S	Menge der Arbeitsstationen	\mathbb{N}
R	Menge an Abhängigkeitsbeziehungen zwischen den Arbeitsschritten	$\mathbb{N} \times \mathbb{N}$
Konstanten		
c	Taktzeit	\mathbb{N}
v_t	Benötigte Zeit für Arbeitsschritt $t \in T$	\mathbb{N}
Variablen		
x_{ts}	Genau dann, wenn $x_{ts} = 1$ wird Arbeitsschritt $t \in T$ der Station $s \in S$ zugeteilt	$\{0, 1\}$

Tabelle 1: Notation Fließbandabstimmung




1.1.2 Anwendungsfall Reihenfolgeplanung (Sequencing)






Wie bereits in Abschnitt 1.1 beschrieben, folgt in modernen Produktionen auf die Fließbandabstimmung, in der die benötigten Arbeitsschritte auf Arbeitsstationen verteilt wurden, häufig die operative Reihenfolge(fein)planung, auch *Sequencing* genannt. Hierbei geht es nun darum, konkrete Arbeitsaufträge, die die einzelnen Stationen durchlaufen müssen, in die optimale Reihenfolge zu bringen. Wir legen hier eine sogenannte *mixed-model assembly line* zugrunde, was bedeutet, dass die Aufträge die Herstellung verschiedener Modelle eines Basisprodukts beschreiben. Die benötigten Arbeitsschritte und die Zeit für deren Erledigung unterscheiden sich somit pro Auftrag. Die operative Feinplanung muss daher u.a. dafür sorgen, dass zu keinem Zeitpunkt zu große Abweichungen (Drift) von den in der Fließbandabstimmung verwendeten Mittelwerten pro Station entstehen.

Eine Problemistanz für das Sequencing besteht aus einem Fließband, einer Liste an Aufträgen und einer Liste an Anforderungen bzw. Regeln für die Reihenfolge. Das Ziel ist es, die Aufträge in eine Reihenfolge zu bringen, die diese Regeln beachtet und eine vorher definierte Zielfunktion optimiert. Im Folgenden werden nun die Eingabeparameter und die verschiedenen Komponenten der Zielfunktion und einer validen Lösung genauer erläutert.

Taktzeit: 10s

Station 1	Station 2	Station 3
Max Drift: 3s	Max Drift: 6s	Max Drift: 2s
Max Takte: 1	Max Takte: 1	Max Takte: 1

Auftragsart	Bearbeitungszeit		
	Stat. 1	Stat. 2	Stat. 3
	13s	12s	10s
	10s	9s	10s
	5s	5s	10s

Auftragsliste:     

(a) Ein Fließband mit drei Stationen, die jeweils aus einem Arbeitsplatz bestehen, einen Takt dauern und eine maximale Driftzeit erlauben. Ein Auftrag an Station 1 muss vollständig abgearbeitet sein, bevor er an Station 2 weiterbearbeitet werden kann (durchgezogene Linie).

(b) Die Aufträge symbolisieren durch die verschiedenen Formen die Herstellung verschiedener Varianten eines Basisprodukts und haben eine individuelle Bearbeitungszeit pro Station. Die zu erledigenden Aufträge sind vorgegeben.

Abbildung 4: Eine einfache Problem Instanz für das Sequencing. Das Ziel ist eine optimale Abarbeitungsreihenfolge für die Auftragsliste zu finden.

Das Fließband ist üblicherweise bereits durch die Fließbandabstimmung (siehe Abschnitt 1.1.1) konfiguriert worden und hat somit eine feste Anzahl an *Arbeitsstationen*—denen die verschiedenen Arbeitsschritte, die an einem Produkt ausgeführt werden müssen, zugeteilt wurden—und eine feste *Taktzeit*. Ein einfaches Beispiel für ein Fließband mit drei Stationen ist in Abb. 4a illustriert.

Fließband:
Taktzeit
Stationen

Jede Station hat eine vordefinierte Anzahl an *Takten*, für die ein Produkt auf dieser Station verweilen kann, und ist selbst wieder in eine Menge an *Arbeitsplätze* aufgeteilt. Jeder Arbeitsplatz kann das Produkt selbst wieder für eine feste Anzahl an Takten bearbeiten (was auch dazu führen kann, dass sich ein Arbeitsplatz über mehrere Stationen erstreckt) und hat eine Liste an Vorgängerarbeitsplätzen, an denen ein Produkt schon vollständig bearbeitet worden sein muss, bevor es an dem aktuellen Arbeitsplatz bearbeitet werden kann.

Station:
Takte
Plätze

Arbeitsplatz:
Takte
Vorgänger
Drifts

Des Weiteren können für jeden Arbeitsplatz erlaubte *Drifts* festgelegt werden: Die Arbeit an diesem Platz darf dann um diese Zeiteinheit früher beginnen bzw. über das Ende der Taktzeit hinausdauern. Bei der ersten Arbeitsstation kann es zudem zu einem initialen Drift kommen, der z.B. durch Verspätungen aus einer vorangegangenen Schicht (mit eigener Reihenfolgeplanung) entstanden ist. Die Driftgrenzen dürfen nie überschritten werden,

da das Produkt dann beispielsweise außerhalb des Arbeitsradius bestimmter Werkzeuge gelangt und das Fließband gestoppt werden müsste. Wenn der maximale Drift an einem Arbeitsplatz überschritten würde, muss daher eine sogenannte *Springerin* eingesetzt werden, die dann pünktlich den Folgeauftrag übernimmt, während die Person an dem Arbeitsplatz ihren aktuellen Auftrag beendet. Diese Person kann dann den Folgeauftrag, der ja bereits von der Springerin bearbeitet wird, überspringen und pünktlich zum Taktschlag mit dem nächsten Auftrag fortfahren.

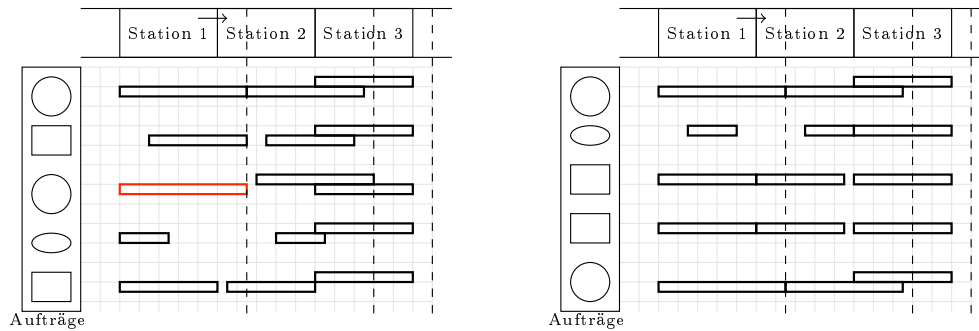
Die Aufträge bestehen aus einer Auflistung verschiedener Arbeitsschritte, die auf den Arbeitsstationen ausgeführt werden müssen, um ein konkretes Produkt herzustellen. Pro Auftrag kennen wir die benötigte *Arbeitszeit* pro Arbeitsplatz und eine Liste von *Attributen*, anhand derer Regeln für die Reihenfolge bestimmt werden können. Ein vereinfachtes Beispiel ohne Attribute ist in Abb. 4b dargestellt.

Auftrag:
 Zeit
 Attribute

Die Regeln schränken den Lösungsraum, also die Anzahl zulässiger Reihenfolgen für die Aufträge, zusätzlich ein. So lässt sich beispielsweise festlegen, dass Aufträge mit einem bestimmten Attribut einen gewissen zeitlichen Mindestabstand haben. Die Regeln stellen jedoch keine starren Vorschriften dar, sondern sollen so gut es geht erfüllt werden.

Sollen
 Attribute
 zunächst
 vernach-
 lässigt
 werden?
 Wenn
 nicht,
 sollte das
 Beispiel
 ggf. er-
 weitert
 werden

Die Lösung einer konkreten Probleminstanz beinhaltet eine hinsichtlich der Zielfunktion optimierte Reihenfolge der Aufträge. Konkret setzt sich der Zielfunktionswert aus drei Komponenten zusammen, die je nach Wichtigkeit mit einem festgelegten Faktor in die Zielfunktion eingehen: Die Anzahl an benötigten Springerinnen, der Drift und die Anzahl an verletzten Regeln. Im einfachen Beispiel (ohne Attribute und Regeln) in Abb. 5a würden dementsprechend die benötigte Springerin und die Drifts an Station 1 und 2 in die Zielfunktion eingehen. Abb. 5b stellt zudem eine Lösung der Instanz dar, die ohne Springerinnen auskommt.



(a) Die hier angegebene (nicht optimale) Lösung erfordert eine Springerin (rot), da der dritte Auftrag sonst den zulässigen maximalen Drift an Station 1 überschritte.

(b) Diese Lösung ist hinsichtlich der Anzahl der Springerinnen und der aufsummierten Drifts optimal.

Abbildung 5: Beispiel für zwei Lösungen der Instanz aus Abb. 4. Die festgelegte Auftragsreihenfolge liest sich von oben nach unten und die Bearbeitungszeiten der Aufträge sind durch die Länge der horizontalen Balken symbolisiert. Zudem markieren die gestrichelten Linien die maximalen Drifts an den Stationen. Da die Arbeit an Station 2 erst nach dem Arbeitsende an Station 1 beginnen kann, kann ein Drift an Station 1 den Bearbeitungsstart an Station 2 verzögern. Der Startzeitpunkt auf Station 2 ergibt sich also aus dem Maximum von dem Ende des Auftrags auf Station 1 und dem Drift des vorherigen Auftrags auf Station 2. Station 3 hingegen hat keine Vorgängerbeziehungen und kann alle Aufträge pünktlich starten, weil sich ein Drift auf Station 2 nicht auf Station 3 auswirkt, sondern die Arbeiten an diesen Stationen gleichzeitig erledigt werden können.

Datensatz

Der Datensatz wurde mittels eines *data generators* künstlich erzeugt. Dafür wurden für die verschiedenen Parameter Werte ausgewählt bzw. ein Bereich angegeben, aus dem der Wert des Parameters gleichverteilt ausgewählt wurde, um Instanzen verschiedener realer Probleme zu simulieren.

Um die Komplexität des Problems zu verringern, wurden in Bezug auf das oben präsentierte Problem folgende Vereinfachungen vorgenommen:

Liste von Vereinfachungen updaten!

- Jede Station besteht aus nur einem Arbeitsplatz, d.h. die Aufteilung einer Arbeitsstation in verschiedene Arbeitsplätze wurde weggelassen.
- ...

Nachdem die Parameter für den Datengenerator festgelegt wurden, sollte hier angegeben werden, wie die Parameter gesetzt wurden und wie viele Instanzen welcher Größe generiert wurden. Vielleicht kann der generierte Datensatz mit Grafiken/Tabellen dargestellt werden (wie viele Instanzen welcher Art, etc.)?!

1.2 Anwendungsfall Testfahrzeugkonfiguration

Bevor ein neues Fahrzeug in Serie produziert werden kann, müssen mit Vorserienfahrzeugen eine bestimmte Anzahl von Tests ausgeführt werden, um die Funktion der einzelnen Komponenten und die allgemeine Produzierbarkeit sicherzustellen. Dabei hat sich die Anzahl der zu testenden Funktionen in den letzten Jahren signifikant gesteigert. Da die Herstellung dieser Vorserienfahrzeuge sehr kostenintensiv und die Ausführung der dazugehörigen Tests sehr zeitaufwendig ist, soll für eine gegebene Anzahl an Tests die Zahl der zu produzierenden Testfahrzeuge minimiert werden.

Jeder Test gehört dabei einer bestimmten Kategorie an, die über die Reihenfolge der Ausführung entscheidet. Crashtests (Z) müssen zum Beispiel immer als letzter Test eines Vorserienautos durchgeführt werden, da diese Teile der Vorserienfahrzeuge zerstören können und somit die Ausführung der anderen Tests unmöglich machen.

Die genauen Zusammenhänge der Testkategorien können den Abb. 6 und 7 entnommen werden.

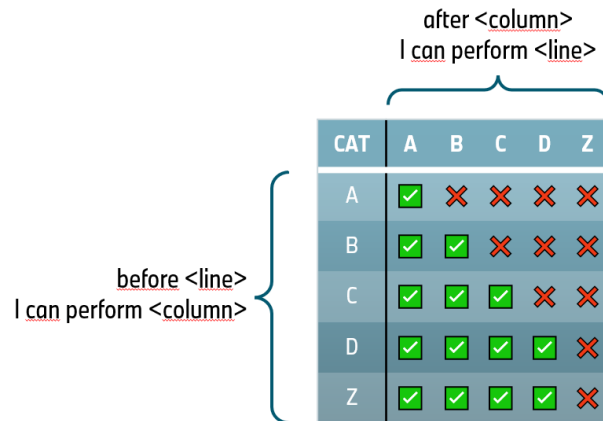


Abbildung 6: Zusammenhang der Testkategorien als Matrix

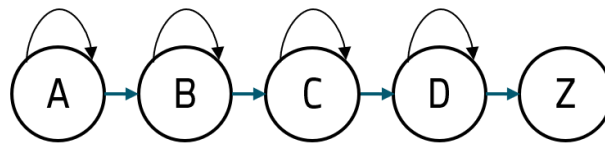


Abbildung 7: Zusammenhang der Testkategorien vereinfacht als gerichteter Graph

Die Testfahrzeuge werden in drei verschiedenen, teilweise überlappenden, Bauphasen produziert, die in Abb. 8 dargestellt sind. Jeder Test ist an Fahrzeuge einer bestimmten Bauphase gekoppelt. Eine Bauphase erstreckt sich über mehr als ein Jahr. Die Tests müssen spätestens ein Jahr nach Bau eines Fahrzeugs absolviert sein.

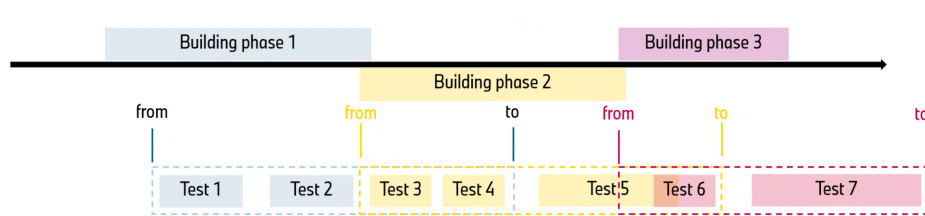


Abbildung 8: Bauphasen der Vorserienfahrzeuge

Pro konkreter Bauphase ist somit eine Menge an Tests vorgegeben, die auf n verschiedenen Testfahrzeugen durchgeführt werden müssen. Pro Test sind dabei gewisse Sonderausstattungen (SAs) vorgegeben oder ausgeschlossen. In Testfahrzeuge können je nach Modelltyp unterschiedliche SAs verbaut werden. Manche SAs können andere SAs fordern oder ausschließen. Zum Beispiel kann ein Auto nicht gleichzeitig einen 4- und 8 Zylindermotor verbaut haben. Diese direkten Abhängigkeiten können auf die Tests übertragen werden, sodass eine minimale Anzahl an Testfahrzeugen gefunden werden kann.

1.2.1 Transformation Testinkompatibilitäten

Ziel ist es herauszufinden, welche Tests auf demselben Testfahrzeug getestet werden dürfen. Dies ergibt sich aus den folgenden Informationen:

- Abhängigkeiten und Ausschlüsse der SAs
- Verfügbare Testfahrzeugkategorien
- Forderung von unterschiedlichen Testfahrzeugen
- Zugehörigkeit zu Testkategorien

Abhängigkeiten und Ausschlüsse der SAs Für jeden Tests ist bekannt, welche SAs des Testfahrzeugs gefordert werden. Neben den direkten Abhängigkeiten gibt es auch die indirekten, die sich aus den geforderten SAs ergeben. Wie die indirekten Abhängigkeiten auf die Tests übertragen werden können, sollen exemplarisch die nachfolgenden Regeln zeigen.

Angenommen eine Sonderausstattung $SA1$ wird gefordert.

Bedingung an SAs	Abhängigkeit im Test mit $SA1$
$SA1 \Rightarrow SA2$	$SA2$ für Test benötigt
$SA1 \Rightarrow \sim SA2$	$SA2$ für Test ausgeschlossen
$SA1 \Rightarrow SA2 \mid SA3$	$SA2$, $SA3$ oder beide für Test benötigt
Nur eins $SA1$, $SA2$, $SA3$	$SA2$ und $SA3$ für Test ausgeschlossen
$SA1 \& SA2$	$SA2$ für Test benötigt
$SA2 \Rightarrow \sim SA1$	dh. $SA1 \Rightarrow \sim SA2$ $SA2$ für Test ausgeschlossen

Wenn eine benötigte SA oder ausgeschlossene $\sim SA$ gefunden wurde, müssen auch für diese die Regeln überprüft werden. So ergibt sich eine Menge an

Tests, für die bekannt ist, welche SAs auf dem Testfahrzeug platziert werden (a) müssen, (b) dürfen und (c) welche nicht platziert werden dürfen.

Tests, die sich ausschließende SAs beinhalten, können z.B. nicht auf demselben Testfahrzeug durchgeführt werden. Sich gegenseitig bedingende SAs müssen hingegen im selben Testfahrzeug verbaut werden.

Verfügbare Testfahrzeugkategorien Des Weiteren sind auch die SAs auf den Testfahrzeugen eingeschränkt. Testfahrzeuge gehören unterschiedlichen Testfahrzeugkategorien an. Jede Testfahrzeugkategorie hat erlaubte SAs.

Vor der Optimierung (als Teil des Pre-Processings) müssen folgende Punkte sichergestellt werden: mindestens eine Testfahrzeugkategorie erlaubt alle benötigten Sonderausstattungen eines Tests. Alle Sonderausstattungen eines Tests müssen miteinander kompatibel sein.

Tests, die nicht auf dem selben Testfahrzeug ausgeführt werden können, können ebenfalls gegenseitig ausgeschlossen werden.

Forderung von unterschiedlichen Testfahrzeugen Es gibt Tests, die mehr als einmal auf unterschiedlichen Testfahrzeugen durchgeführt werden müssen. Diese Tests schließen sich somit auch gegenseitig aus.

Zugehörigkeit zu Testkategorien Aus den Testfahrzeugkategorien ergibt sich die Ausführungsreihenfolge der Tests. Tests, die zur Kategorie 'Z' gehören, dürfen allerdings nur einmal pro Fahrzeug existieren. Diese Tests sind somit gegenseitig auszuschließen, damit gewährleistet werden kann, dass nur ein Test pro Fahrzeug zugeteilt werden kann. Tests anderer Kategorien müssen nicht besonders behandelt werden.

1.2.2 Transformation zu Bin Packing mit Konflikten

Gegeben ist eine Liste an Tests einer konkreten Bauphase, die genau einmal durchgeführt werden müssen. Für jeden Test ist bekannt

- welche Tests nicht am selben Fahrzeug durchgeführt werden dürfen,
- wie viel Zeit für die Durchführung benötigt wird,
- wann er frühestmöglich ausgeführt werden darf und

- bis wann er spätestens ausgeführt werden muss.

Um die Kosten möglichst gering zu halten, soll die Anzahl der Testfahrzeuge minimiert werden. Alle Tests müssen ausgeführt werden.

Das Testfahrzeugkonfigurationsproblem lässt sich auf das spezielle Bin Packing Problem mit Konflikten (siehe Abschnitt 2.1.2) abbilden.

Die Testfahrzeuge sind dabei die Behälter, deren Anzahl im Optimum möglichst gering sein muss. Die Objekte sind die auszuführenden Tests, die in dieser Bauphase abgeschlossen sein müssen. Hierbei gehen wir davon aus, dass jeder Test genau einer Bauphase zugeordnet ist. Die maximale Kapazität jedes Behälters ist eine Randbedingung des Optimierungsproblems und zunächst auf ein Jahr beschränkt. Die Test-Durchführungszeit ist in der klassischen Problemformulierung das Gewicht der Objekte. Alle einem Testfahrzeug zugewiesenen Tests müssen sequenziell innerhalb des Testzeitraums durchgeführt werden, der der maximalen Kapazität eines Behälters entspricht.

Die letzten beiden Eigenschaften der Tests bezüglich der Ausführungszeit ignorieren wir zunächst. Man könnte das Ergebnis des Bin Packing Problems als Grundlage für das Scheduling nutzen, indem man die durchzuführenden Tests unter Berücksichtigung der Testkategorien sortiert.

Zusammengefasst, lässt sich die Testfahrzeugkonfiguration wie folgt auf die Notation aus Abschnitt 2.1.2 mappen:

Marvin:
Check
Da-
ten-
satz,
jeder
Test
nur
eine
Bau-
phase?

Marvin:
Check
Da-
ten-
satz,
Tests
se-
quen-
ziell?

Symbol	Bedeutung	Definitionsbereich
Mengen		
O	Menge der Tests	\mathbb{N}
B	Menge der Testfahrzeuge	\mathbb{N}
I	Menge der inkompatiblen Tests	$\mathbb{N} \times \mathbb{N}$
Konstanten		
c	Ein Jahr	\mathbb{N}
w_o	Benötigte Zeit für Test $o \in O$	\mathbb{N}
Variablen		
x_{ob}	Genau dann, wenn $x_{ob} = 1$ wird Test $o \in O$ dem Testfahrzeug $b \in B$ zugeteilt	$\{0, 1\}$

Tabelle 2: Notation zu Testfahrzeugkonfiguration

2 Mathematische Grundprobleme

Im Folgenden sollen die mathematischen Grundprobleme analysiert werden.

2.1 Bin Packing

Beim Bin Packing Problem (oder auch Behälterproblem) geht es darum, eine gegebene Anzahl an Objekten auf eine kleinstmögliche Anzahl an Behältern zu verteilen, sodass kein Behälter überläuft.

Als ganzzahliges Optimierungsproblem lässt sich das Bin Packing Problem wie folgt darstellen: Jedes Objekt muss genau einem Behälter zugeteilt werden (2). Pro Behälter darf die maximale Kapazität nicht überschritten werden (3). Das Ziel ist, die Anzahl der Behälter zu minimieren, was mithilfe der Zielfunktion (1) erreicht wird.

Marvin:
Check
Da-
ten-
satz,
fehlt
ir-
gend-
was?

Symbol	Bedeutung	Definitionsbereich
Mengen		
O	Menge der Objekte	\mathbb{N}
B	Menge der Behälter	\mathbb{N}
Konstanten		
c	Maximale Kapazität der Behälter	\mathbb{N}
w_o	Gewicht von Objekt $o \in O$	\mathbb{N}
Variablen		
x_{ob}	$x_{ob} = 1$ gdw. Objekt $o \in O$ dem Behälter $b \in B$ zugeteilt wird	$\{0, 1\}$
y_b	$y_b = 1$ gdw. Behälter $b \in B$ verwendet wird	$\{0, 1\}$

Tabelle 3: Notation Bin Packing

$$\min \sum_{b \in B} y_b \quad (1)$$

$$s.t. \quad \sum_{b \in B} x_{ob} = 1, \quad \forall o \in O \quad (2)$$

$$\sum_{o \in O} w_o \cdot x_{ob} \leq c \cdot y_b, \quad \forall b \in B \quad (3)$$

$$x_{ob}, y_b \in \{0, 1\}, \quad \forall o \in O, b \in B \quad (4)$$

Alternative Zielfunktionen. In einer früheren Version dieses Dokuments wurde als Zielfunktion statt (1)

$$\sum_{b \in B, o \in O} b \cdot x_{ob}$$

minimiert,¹ was auch in der Literatur (für das speziellere Assembly Line Balancing Problem) zu finden ist [alb_paper_50]. Diese Version ermöglicht im Vergleich zu der obigen Formulierung zwar die y Variablen einzusparen,

¹Die Constraints können fast unverändert bleiben; es müssen lediglich die y Variablen gelöscht werden.

führt aufgrund der Gewichtung allerdings manchmal zu suboptimalen Lösungen in Bezug auf die Anzahl der benötigten Bins.

Um dieses Problem zu vermeiden, kann in der Zielfunktion mit einer *cost explosion* gearbeitet werden, um zusätzliche Bins stärker zu bestrafen:

$$\min \sum_{b \in B, o \in O} |O|^b \cdot x_{ob}$$

Der Exponent kann auch $b - 1$ sein, wenn Bin 0 den Koeffizienten 0 hat.

Da die Koeffizienten, abhängig von $|B|$, sehr groß werden können, kommt es in der Praxis jedoch schnell zu numerischen Problemen, weshalb diese Variante hier nur von theoretischem Interesse ist.

2.1.1 Assembly Line Balancing Problem

Eine interessante Erweiterung des Bin Packing Problems ergibt sich durch die Einführung von Reihenfolgebeziehungen zwischen den Objekten.

In der Literatur ist das Problem auch als Simple Assembly Line Balancing Problem (SALBP) bekannt. Bei diesem geht es darum, eine Reihe von Aufgaben mit Vorrangbedingungen auf mehrere hintereinanderliegende Stationen zu verteilen [alb_paper_50]. Bei der im Folgenden betrachteten Problemvariante SALBP-1 ist die maximale Zeit für die Ausführung aller Aufgaben, die einer Station zugeteilt werden, vorgegeben, während die Anzahl der Stationen minimiert werden soll.

Die Objekte des Bin Packing Problems sind somit nun die Aufgaben und die Behälter die Stationen des Problems.

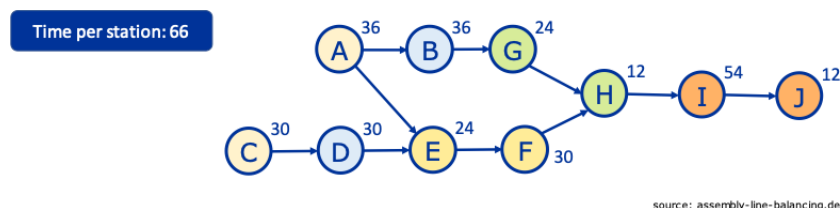


Abbildung 9: Beispiel Assembly Line Balancing Problem und Lösung

Als ganzzahliges Optimierungsproblem lässt sich SALBP-1 wie folgt darstellen: Jede Aufgabe muss genau einer Station zugeteilt werden (6). Pro Station muss die maximale Bearbeitungszeit eingehalten werden (7). Die Reihenfolgebedingungen müssen eingehalten werden. Das heißt, wenn zwischen zwei Aufgaben eine vorgegebene Reihenfolge gilt, dann darf die zweite Aufgabe nicht auf einer Station durchgeführt werden, die vor der ersten Aufgabe liegt (8). Das Ziel ist, die Anzahl der benötigten Stationen zu minimieren, was mithilfe der Zielfunktion (5) erreicht wird.

Symbol	Bedeutung	Definitionsbereich
Mengen		
T	Menge der Aufgaben	\mathbb{N}
S	Menge der Stationen	\mathbb{N}
R	Menge an Vorrangbedingungen zwischen zwei Aufgaben	$\mathbb{N} \times \mathbb{N}$
Konstanten		
c	Taktzeit	\mathbb{N}
v_t	Benötigte Zeit für Aufgabe $t \in T$	\mathbb{N}
Variablen		
x_{ts}	$x_{ts} = 1$ gdw. Aufgabe $t \in T$ Station $s \in S$ zugeteilt wird	$\{0, 1\}$
y_s	$y_s = 1$ gdw. Station $s \in S$ verwendet wird	$\{0, 1\}$

Tabelle 4: Notation SALBP-1

$$\min \quad \sum_{s \in S} y_s \quad (5)$$

$$s.t. \quad \sum_{s \in S} x_{ts} = 1, \quad \forall t \in T \quad (6)$$

$$\sum_{t \in T} v_t \cdot x_{ts} \leq c \cdot y_s, \quad \forall s \in S \quad (7)$$

$$\sum_{s \in S} s \cdot x_{ts} \leq \sum_{s \in S} s \cdot x_{t's}, \quad \forall (t, t') \in R \quad (8)$$

$$x_{ts}, y_s \in \{0, 1\}, \quad \forall t \in T, s \in S \quad (9)$$

Alternative Zielfunktionen. Neben den alternativen Zielfunktionen, die bereits in Abschnitt 2.1 besprochen wurden, gibt es für SALBP-1 eine weitere Möglichkeit (cf.[patterson1975assembly]): Falls es in dem durch R induzierten Reihenfolgegraphen mehrere Senken gibt, wird eine fiktive Aufgabe t_s eingeführt, die nach allen anderen Aufgaben ausgeführt werden muss (R muss also für alle ursprünglichen Senken t um (t, t_s) erweitert werden). Da t_s offensichtlich auf der letzten Station ausgeführt werden muss, kann in der Zielfunktion die Stationsnummer dieser Aufgabe minimiert werden, um die Gesamtzahl an Stationen zu minimieren:

$$\min \sum_{s \in S} s \cdot x_{st_s}$$

Im Hinblick auf die Minimierung der benötigten Variablen und Constraints könnte diese Formulierung dann hilfreich sein, wenn es in einer konkreten SALBP-1 Instanz bereits wenig Senken im Reihenfolgegraphen gibt und potentiell viele Stationen benötigt werden (wodurch die ursprüngliche Formulierung viele y Variablen erfordern würde).

2.1.2 Bin Packing mit Konflikten

Eine weitere interessante Erweiterung des Bin Packing Problems ergibt sich durch die Einführung von inkompatiblen Objekten. Inkompatible Objekte müssen zwingend auf unterschiedliche Behälter aufgeteilt werden (13).

Dadurch ergibt sich das folgende ganzzahlige Optimierungsproblem:

Symbol	Bedeutung	Definitionsbereich
Mengen		
O	Menge der Objekte	\mathbb{N}
B	Menge der Behälter	\mathbb{N}
I	Menge der inkompatiblen Objekte	$\mathbb{N} \times \mathbb{N}$
Konstanten		
c	Maximale Kapazität der Behälter	\mathbb{N}
w_o	Benötigte Größe für Objekt $o \in O$	\mathbb{N}
Variablen		
x_{ob}	Genau dann, wenn $x_{ob} = 1$ wird Objekt $o \in O$ dem Behälter $b \in B$ zugeteilt	$\{0, 1\}$
y_b	$y_b = 1$ gdw. Behälter $b \in B$ verwendet wird	$\{0, 1\}$

Tabelle 5: Notation zu Bin Packing mit Konflikten

$$\min \quad \sum_{b \in B} y_b \quad (10)$$

$$s.t. \quad \sum_{b \in B} x_{ob} = 1, \quad \forall o \in O \quad (11)$$

$$\sum_{o \in O} w_o \cdot x_{ob} \leq c \cdot y_b, \quad \forall b \in B \quad (12)$$

$$x_{ob} + x_{o'b} \leq 1, \quad \forall (o, o') \in I, b \in B \quad (13)$$

$$x_{ob} \in \{0, 1\}, \quad \forall o \in O, b \in B \quad (14)$$

3 Problem mappings

3.1 Bin packing with conflicts - QUBO model

Consider the bin packing IP model from Abschnitt 2.1.2.

Define the QUBO cost function as

$$H(x, y) = A \cdot H_A(x, y) + B \cdot H_B(x) \quad (15)$$

The cost term H_B models the costs in eq. (10):

$$H_B(x) = \sum_{b \in B, o \in O} b \cdot x_{ob} . \quad (16)$$

The penalty term H_A encodes feasibility.

For every constraint in the IP model, a penalty term for its violation is introduced. For the bin-assignment-constraints (11):

$$\sum_{o \in O} \left(\sum_{b \in B} x_{ob} - 1 \right)^2 . \quad (17)$$

For the object-conflict-constraints (13) we introduce:

$$\sum_{b \in B} \sum_{o, o' \in I} (x_{ob} \cdot x_{o'b}) . \quad (18)$$

For the bin-capacity-constraints (12), auxiliary variables y are necessary. For $b \in B$, the *slack* is

$$S_b := c - \sum_{o \in O} w_o \cdot x_{ob} \geq 0 . \quad (19)$$

Unary (one-hot) encoding of slack. We introduce $y_{sb} \in \{0, 1\}$, $b \in B$, $s \in \{0, 1, \dots, c\}$, where $y_{sb} = 1 \Leftrightarrow S_b = s$. Thus,

$$S_b = \sum_{s=0}^c s \cdot y_{sb} \quad \forall b \in B . \quad (20)$$

This is modeled by introducing two additional penalty terms into H_A . First, for each $b \in B$, only a single y_{sb} is equal to one:

$$\sum_{b \in B} \left(1 - \sum_{s=1}^c y_{sb} \right)^2 . \quad (21)$$

Second, for each $b \in B$, negative slack is penalized:

$$\sum_{b \in B} \left(\sum_{s=0}^c s \cdot y_{sb} - \sum_{o \in O} w_o \cdot x_{ob} \right)^2 . \quad (22)$$

In total, we need $|B| \cdot (c + 1)$ additional variables.

Binary (logarithmic) encoding of slack. Here, we introduce a different slack-encoding with $|B| \cdot \lceil \log_2(c+1) \rceil$ additional variables. We introduce $y_{kb} \in \{0, 1\}$, $b \in B$, $k \in \{0, 1, \dots, M\}$, where

$$M := \lceil \log_2(c+1) \rceil. \quad (23)$$

Then, the slack has a binary representation

$$S_b = \sum_{k=0}^{M-2} 2^k \cdot y_{kb} + (c - 2^{M-1} + 1) \cdot y_{M-1,b}. \quad (24)$$

We add a penalty to H_A if the slack does not have such a binary representation, i.e. Constraint (12) is violated:

$$\sum_{b \in B} \left(\sum_{k=0}^{M-2} 2^k \cdot y_{kb} + (c - 2^{M-1} + 1) \cdot y_{M-1,b} - \sum_{o \in O} w_o \cdot x_{ob} \right)^2. \quad (25)$$

Now we can get $H_A(x, y)$ by adding up

4 Metriken

4.1 Berechnungszeit

Die erste wichtige Metrik beim Benchmarking von Algorithmen für Quantencomputer zum Lösen von kombinatorischen Optimierungsproblemen ist die Zeit, die für die Ausführung des Algorithmus benötigt wurde. Hierbei wird nicht das Generieren einer Lösung, noch die Qualität einer solchen Lösung bewertet, sondern lediglich die Zeit gemessen, die während der Ausführung des Algorithmus vergangen ist. Außerdem ist es bei mehrgliedrigen Algorithmen sinnvoll, die Berechnungszeit der einzelnen Teile zu erfassen. Beispielsweise sollte man bei der Ausführung der Quantum Approximate Optimization Algorithm (QAOA) die Parameter-Training-Zeit als auch die Ausführzeit erfassen.

4.2 Lösungsqualität

Die zweite wegweisende Metrik ist die Qualität der generierten Lösung. Bei Optimierungsproblemen ist natürlich das Ziel, die optimale Lösung mit dem

Algorithmus zu generieren. Aber auch schon das Finden von sehr guten zulässigen, aber nicht optimalen Lösungen kann sehr wertvoll sein. Vor allem bei sehr komplexen Optimierungsproblemen, bei denen die Rechenleistung eines klassischen Computers für eine optimale Lösung nicht ausreicht, können Quantencomputer eine sehr gute Alternative darstellen, um in wenig Rechenzeit gute zulässige Lösungen zu erzeugen. Folglich ist das erste wichtige Kriterium für die Lösungsqualität die Zulässigkeit der durch den Quantencomputer generierten Lösung. Wenn die optimale Lösung eines Problems bekannt ist, kann die Qualität der zulässigen Lösung des Quantencomputers quantifiziert werden, durch die Differenz der Zielfunktionswerte der optimalen Lösung und der Quantencomputer-Lösung. Eine solche optimale Lösung kann generell durch einen klassischen Computer bestimmt werden. Jedoch kommen klassische Computer bei sehr komplexen Problemen an ihre Grenzen. In einem solchen Fall, bei dem die Rechenleistung eines klassischen Computers nicht ausreicht, ist die optimale Lösung nicht bekannt und es muss eine andere Möglichkeit gefunden werden, um die Lösungsqualität zu quantifizieren. Optionen hierfür wären beispielsweise:

- Differenz der Zielfunktionswerte der besten Quantencomputerlösungen mit der von einem klassischen Algorithmus in der gleichen Rechenzeit bestimmten besten Lösung.
- Vergleich der Zielfunktionswerte der besten Quantencomputerlösungen mit Random-Sampling

4.3 Kombinationen der Metriken

4.3.1 Time-to-solution

Vorgeschlagen in "Application-Oriented Performance Benchmarks for Quantum Computing" (Sankar, Scherer, Kako), indiziert die "Time-to-solution-Metrik sowohl die Berechnungszeit als auch die Lösungsqualität. Hier wird die time-to-solution t_s berechnet durch:

$$t_s = R_{99} t_{max} \quad , \quad (26)$$

wobei $R_{99} = \frac{\log(0.01)}{\log(1-P_s)}$ die Anzahl der Shots ist, die für eine 99% Wahrscheinlichkeit [\[https://arxiv.org/abs/2110.03137\]](https://arxiv.org/abs/2110.03137) die optimale Lösung gefunden wird.

4.3.2 Q-Pack

Eine Benchmarking-Metrik für den QAOA, die folgende Aspekte berücksichtigt:

- die maximale Problemgröße, die der QC lösen kann (\Rightarrow Grenzen für das Optimierungsproblem und die QAOA-Layer-Anzahl p)
- Berechnungszeit (\Rightarrow Overall, Klassischer Computer, Dauer der Connection zum QC, Dauer des Preprocessings für den QC, Dauer des QC-Queueings, QC-Runtime)
- die erreichte Lösungsgenauigkeit (Accuracy) (\Rightarrow Entfernung der QC-Lösung von der optimalen Lösung)

QPack ist auf folgende Optimierungsprobleme zugeschnitten:

- Max-Cut
- dominating set
- travelling salesman problem

[<https://arxiv.org/abs/2103.17193>] Der wesentliche Aspekt, der bei Q-Pack dazukommt, ist die Scalability, also die zusätzliche Auskunft über die Performance bei verschiedenen Problemgrößen.

4.4 Energie-verbrauch / -kosten

Die bei der Ausführung auf dem Quantencomputer verbrauchte Energie ist ebenfalls eine interessante Metrik. Jedoch ist dies schwer herauszufinden. (siehe Abbas, <https://arxiv.org/abs/2312.02279>)

4.5 Hardware-spezifische Metriken

4.5.1 Quantum Volume

Von IBM Quantum. [<https://quantum-computing.ibm.com>, 2021. accessed 2021-05-15] vorgestellt, ist das "Quantum Volume" genau 2^n , wobei n die Länge als auch die Höhe des größten Quanten-Schaltkreises ist, der mit mehr als einer $\frac{2}{3}$ -Wahrscheinlichkeit erfolgreich ausgeführt wird. Beispielsweise, wenn ein QC einen Schaltkreis mit Länge und Höhe 5 erfolgreich ausführt, aber nicht mehr mit Länge und Höhe 6, dann ist das $QV(QC) = 2^5 = 32$

4.5.2 Gate Error Rates

Es ist wichtig, bei der Performance eines gatebasierten QC-Algorithmus auch die Fehlerraten zu betrachten.

5 Versionshistorie

Version	Datum	Autor(en)	Änderungen
0.0	2023-07-28	OptWare	Initiale Version: Beschreibung Use-Case, Bin Packing
0.1	2023-09-05	OptWare	Anpassung Beschreibung Testfahrzeugkonfiguration
0.2	2023-09-20	OptWare	Anpassung der Zielfunktionen
0.3	2023-12-19	OptWare	Beschreibung des Anwendungsfalls - Reihenfolgeplanung
0.4	2024-03-12	Fraunhofer IIS	Mapping des Bin Packing Problems zu QUBO Formulierung

Liste der noch zu erledigenden Punkte

Sollen Attribute zunächst vernachlässigt werden? Wenn nicht, sollte das Beispiel ggf. erweitert werden	8
Liste von Vereinfachungen updaten!	9
Nachdem die Parameter für den Datengenerator festgelegt wurden, sollte hier angegeben werden, wie die Parameter gesetzt wurden und wie viele Instanzen welcher Größe generiert wurden. Vielleicht kann der generierte Datensatz mit Grafiken/Tabellen dargestellt werden (wie viele Instanzen welcher Art, etc.)?!	10
Liste ist nicht vollständig!	12
Marvin: Check Datensatz, exklusives ODER, weitere Abhängigkeiten, die man darstellen müsste?	12
Marvin: Check Datensatz, jeder Test nur eine Bauphase?	14
Marvin: Check Datensatz, Tests sequenziell?	14
Marvin: Check Datensatz, fehlt irgendwas?	15
Der Exponent kann auch $b - 1$ sein, wenn Bin 0 den Koeffizienten 0 hat.	17