Εργασία στο μάθημα
Φυσική Πολλών Σωμάτων και Κβαντικοί Υπολογιστές
ΣΗΜΜΥ 9ο εξάμηνο
Διδάσκων: Βαρελογιάννης Γεώργιος

Battery Revenue Optimization με προσέγγιση QAOA

Κύκλωμα και Βελτιώσεις

Παγώνης Αλέξανδρος
03116076

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο
10 - Δεκεμβρίου - 2021

Battery Revenue Optimization με προσέγγιση QAOA

Κύκλωμα και Βελτιώσεις

Περίληψη

Στην παρούσα εργασία χρησιμοποιούμε την τεχνική QAOA για την επίλυση του προβλήματος Battery Revenue Optimization.

Αρχικά παρουσιάζουμε το πρόβλημα Battery Revenue Optimization και τον αλγόριθμο QAOA. Ύστερα θέτουμε τις παραμέτρους β και γ των μοναδιαιών τελεστών του QAOA, διατυπώνουμε την αντικειμενική συνάρτηση προς μεγιστοποίηση και κατασκευάζουμε το χβαντικό κύκλωμα που την υλοποιεί [1].

Επίσης παρουσιάζουμε μια σειρά από βελτιώσεις για το κύκλωμα οι οποίες μειώνουν τον χρόνο και το κόστος του κυκλώματος, ενώ παράλληλα αυξάνουν την ακρίβεια του αλγορίθμου.Έπειτα το εκτελούμε στον προσομοιωτή της IBM.

Περιεχόμενα

1 Εισαγωγή

1.1 Το πρόβλημα Battery Revenue Optimization

Το Battery Revenue Optimization είναι ένα πρόβλημα βελτιστοποίησης υπό περιορισμό και η διατύπωσή του είναι η εξής: Μια εταιρία διαθέτει μια μπαταρία την οποία θέλει να νοικιάσει προς χρήση. Δύο αγορές $M1$ και $M2$ προτίθενται να την νοικιάσουν για n μέρες, όμως η εταιρία έχει μία μπαταρία οπότε πρέπει να διαλέξει σε ποιά από τις δύο αγορές θα την προσφέρει κάθε μέρα. Η αγορά $M1$ την t-οστή ημέρα θα την νοικιάσει έναντι $\lambda_1^{(t)}$ ευρώ, όμως θα την ζημιώσει κατά $c_1^{(t)}$. Αντίστοιχα, η αγορά $M2$ την t-οστή ημέρα θα την νοικιάσει έναντι $\lambda_2^{(t)}$ ευρώ, όμως θα την ζημιώσει κατά $c_2^{(t)}$. Η μπαταρία έχει αντοχή C_{max} και σκοπός μας είναι να μεγιστοποιήσουμε τα κέρδη της εταιρίας, χωρίς να χρειαστεί δεύτερη μπαταρία. Τυπικά το πρόβλημα έχει τον εξής φορμαλισμό: Για μια επιλογή $z = z_1 z_2 \cdots z_n$, (όπου για $z_t = 0$ επιλέγω την αγορά $M1$ την ημέρα t και για $z_t = 1$ επιλέγω την αγορά $M2$ την ημέρα t) θέλουμε:

$$\begin{aligned} &\underset{z_i \in \{0,1\}^n}{\operatorname{argmax}} \left(\sum_{t=1}^{n} \left[(1 - z_i)\lambda_1^{(t)} + z_i \lambda_2^{(t)} \right] \right) \\ &\tau.\omega. \sum_{t=1}^{n} \left[(1 - z_i)c_1^{(t)} + z_i c_2^{(t)} \right] \leq C_{max} \end{aligned} \tag{1}$$

Για κάθε ημέρα t έχω:

- Αν $\lambda_1^{(t)} \geq \lambda_2^{(t)}$ και $c_1^{(t)} \leq c_2^{(t)}$, τότε επιλέγω την αγορά $M1$
- Αν $\lambda_2^{(t)} \geq \lambda_1^{(t)}$ και $c_2^{(t)} \leq c_1^{(t)}$, τότε επιλέγω την αγορά $M2$
- Αν $\lambda_2^{(t)} \geq \lambda_1^{(t)}$ και $c_2^{(t)} \geq c_1^{(t)}$, τότε έχω κέρδη τουλάχιστον $\lambda_1^{(t)}$ και φθορά τουλάχιστον $c_1^{(t)}$. Οπότε με ενδιαφέρει απλώς αν θα προτιμήσω την αγορά $M2$ με καθαρό κέρδος $\lambda_2^{(t)} - \lambda_1^{(t)}$ και φθορά $c_2^{(t)} - c_1^{(t)}$.
- Αν $\lambda_1^{(t)} \geq \lambda_2^{(t)}$ και $c_1^{(t)} \geq c_2^{(t)}$, τότε έχω κέρδη τουλάχιστον $\lambda_2^{(t)}$ και φθορά τουλάχιστον $c_2^{(t)}$. Οπότε με ενδιαφέρει απλώς αν θα προτιμήσω την αγορά $M1$ με καθαρό κέρδος $\lambda_1^{(t)} - \lambda_2^{(t)}$ και φθορά $c_1^{(t)} - c_2^{(t)}$.

Οι περιπτώσεις 1 και 2 είναι τετριμμένες, ενώ οι 3 και 4 είναι συμμετρικές. Οπότε, χωρίς βλάβη της γενικότητας, μπορώ να υποθέσω οτι βρίσκομαι στην περίπτωση 3. Το πρόβλημα με αυτόν τον φορμαλισμό είναι γνωστό και ως πρόβλημα 0-1 KnapSack, το οποίο θα εξετάσουμε στο κομμάτι των βελτιώσεων. Αυτό που μας ενδιαφέρει τώρα είναι η πολυπλοκότητα του Battery Revenue Optimization Problem από κλασικό αλγόριθμο (και επειδή το Battery Revenue Optimization Problem ανάγεται στο 0-1 KnapSack έχουν ίδια πολυπλοκότητα).

Το πρόβλημα **0-1 KnapSack** είναι **NP-Complete**. Εμείς θα χαλαρώσουμε τις απαιτήσεις του προβλήματος και θα δεχτούμε προσεγγιστικές λύσεις, δηλαδή: λύσεις που ξεπερνάνε το C_{max} (τις οποίες θα απορρίπτουμε) και λύσεις που προσεγγίζουν τα βέλτιστα κέρδη. Οπότε η αναγωγή που μας

ενδιαφέρει είναι στο Approximate 0-1 KnapSack, το οποίο έχει λύσεις από κλασικούς προσεγγιστικούς αλγορίθμους, με τον καλύτερο FPTAS να έχει χρόνο χρόνο [2]:

$$O\left(n\log\left(\frac{1}{\epsilon}\right)+\frac{\left(\frac{1}{\epsilon}\right)^{\frac{9}{4}}}{2^{\Omega\left(\sqrt{\log\left(\frac{1}{\epsilon}\right)}\right)}}\right) \tag{2}$$

Αυτό το αποτέλεσμα δεν είναι και πολύ ενθαρυντικό, αφού θέλουμε η προσέγγιση $1+\varepsilon$ να είναι αρκετά μικρή, οπότε το $\log(\frac{1}{\epsilon})$ είναι αρκετά μεγάλο.

Οι εφαρμογές του Battery Revenue Problem (ή του ισοδυνάμου KnapSack) είναι πολλές. Παραδείγματος χάριν:

- Στην επιλογή των μετοχών που μεγιστοποιούν τα μέσα κέρδη του επενδυτή υπό τον περιορισμό ενός μέγιστου ρίσκου
- Στις τηλεπικοινωνίες, όταν ο αποστολέας επιλέγει μία συμπίεση ή έναν καταμερισμό δεδομένων, ώστε να χωράνε όλα στο μεγαλύτερο πακέτο μεταφοράς δεδομένων

Για ένα τόσο σημαντικό και δύσκολο πρόβλημα, με τέτοιο εύρος εφαρμογών, θα θέλαμε να βρούμε μια καλύτερη λύση, οπότε προσφεύγουμε στον κβαντικό υπολογισμό.

1.2 Adiabatic Computing?

Μια αρχική ιδέα για την λύση του Battery Revenue Optimization Problem είναι η προσέγγιση Quantum Adiabatic Computing. Σε αυτή τη μέθοδο κατασκευάζουμε την χαμιλτονιανή:

$$\hat{H}(t)=(1-t)\,\hat{H}_i+t\cdot\hat{H}_f,\quad t\in[0,1] \tag{3}$$

όπου $\hat{H}_i$ η αρχική κατάσταση και $\hat{H}_f$ η επιθυμητή τελική κατάσταση του συστήματος με ιδιοκαταστάσεις τις λύσεις του προβλήματός μας. Και η ιδέα είναι οτι εν ευθέτω χρόνω το σύστημα θα μεταβεί στην $\hat{H}_f$, οπότε θα έχουμε το αποτέλεσμα. Ωστόσο, η τεχνική αυτή αποτυγχάνει να λύσει το πρόβλημα KnapSack, όπως έχουν δείξει οι προσπάθειες [3]. Επίσης, η χαμιλτονιανή αυτή είναι πολύ δύσκολο να κατασκευαστεί [4]. Αν και η τεχνική αυτή απορρίπτεται, η βασική της ιδέα θα παίξει σημαντικό ρόλο στην επιλογή των παραμέτρων β,γ του QAOA.

1.3 Quantum Approximate Optimization Algorithm

Ο αλγόριθμος Quantum Approximate Optimization Algorithm είναι ένας κβαντικός προσεγγιστικός αλγόριθμος για την λύση μιας οικογένειας προβλημάτων. Συγκεκριμένα, αντιστοιχεί σε ένα διμερές κύκλωμα το οποίο επαναλαμβάνουμε p φορές με σκοπό να προσεγγίσουμε την βέλτιστη λύση. Η ιδέα είναι η εξής:

- Διατυπώνουμε την αντικειμενική συνάτηση $f(z)$ που προσπαθούμε να μεγιστοποιήσουμε. Δηλαδή ψάχνουμε το $z = \underset{z_t \in \{0,1\}^n}{\operatorname{argmax}} f(z)$, όπου $z_t \in \{0,1\}$ οι n επιλογές που πρέπει να κάνουμε.
- Βρίσκουμε τον τελεστή C τέτοιον που $C|z\rangle = f(z)|z\rangle$ και κατασκευάζουμε τον αντίστοιχο μοναδιαίο τελεστή: $U(C,\gamma) = e^{-i\gamma C}$ με την γωνία γ να είναι ένας πολλαπλασιαστικός παράγοντας $\gamma \in (0, 2\pi)$.
- Αν είναι σ_t^x ο τελεστής $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ εφαρμοζόμενος στο t-οστό qubit, ορίζω τον τελεστή B ως το άθροισμα όλων των σ^x: $B = \sum_{t=1}^{n} \sigma_t^x$ και τον αντίστοιχο μοναδιαίο τελεστή $U(B,\beta) = e^{-i\beta B}$ με $\beta \in (0, \pi)$.

Ο τελεστής C υπολογίζει την $f(z)$ όταν εφαρμοστεί στην κατάσταση $|z\rangle$ και ο τελεστής B αναμειγνύει τα qubits εναλλάσοντας τα 0 σε 1 και ανάποδα. Δηλαδή, "δοκιμάζουμε" μία-μία διάφορες υποψήφιες λύσεις (με τον τελεστή C) και "βαθμολογούμε" (με τον τελεστή B). Αρχικά φτιάχνουμε την κατάσταση $|+\rangle^{\otimes n}$ με πύλες Hadamard και μετά εφαρμόζουμε τα κυκλώματα των τελεστών $U(C,\gamma)$ και $U(B,\beta)$ p-φορές στην σειρά με διαφορετικές γωνίες (β,γ). Η τελική κατάσταση, λοιπόν, είναι η:

$$|\beta,\gamma\rangle \equiv U(B,\beta_p)U(C,\gamma_p)\cdots U(B,\beta_1)U(C,\gamma_1)|+\rangle^{\otimes n} \tag{4}$$

Η αναμενόμενη τιμή της C στο τέλος είναι η $F_p(\beta,\gamma) \equiv \langle\beta,\gamma| C |\beta,\gamma\rangle$. Από το κβαντικό αδιαβατικό θεώρημα έχουμε:

$$\lim_{p\to\infty}\left[\max_{(\beta,\gamma)} F_p(\beta,\gamma)\right] = \max_{z\in\{0,1\}^n} C(z) \tag{5}$$

δηλαδή για άπειρες επαναλήψεις p και σωστές παραμέτρους β,γ ο αλγόριθμος θα μας δώσει την βέλτιση λύση $\max_{z\in\{0,1\}^n} C(z)$. Στην πραγματικότητα το p είναι πεπερασμένο, και για σωστά επιλεγμένες παραμέτρους (β,γ) όσο το p αυξάνει, τόσο προσεγγίζω την λύση μου.

Η τεχνική QAOA είναι αρκετα χρήσιμη για πολλά προβλήματα συνδυαστικής βελτιστοποίησης, όπως το MAX-CUT, το KnapSack και το MAX3-SAT [5] (Όλα τους NP-Complete στην εύρεση ακριβούς λύσης και αρκετά δύσκολα στην εύρεση προσεγγιστικής λύσης).

Επιλέγουμε λοιπόν την προσέγγιση QAOA για την λύση του προβλήματος Battery Revenue Optimization. Τα κύρια πλεονεκτήματα έναντι του κλασικού αλγόριθμου είναι η **απλότητα**, η μικρή **χρονική πολυπλοκότητα** $O(pn\log_2(n))$ ή και $O(p(\log_2 n)^3)$ με την βοήθεια $O(n\log_2(n))$ βοηθητικών qubits [1] και η **ευελιξία** (όπως θα δούμε στο κομμάτι των βελτιώσεων). Χρειάζεται να υπολογίσουμε τους τελεστές $U(C,\gamma)$, $U(B,\beta)$ και να κατασκευάσουμε το κύκλωμα που τους υλοποιεί.

2 Κύκλωμα

Για να οριστεί ένα πρόβλημα QAOA μένει να κατασκευάσουμε τα κυκλώματα U(C, γ) και U(B, β). Η βέλτιστη επιλογή των β και γ δεν είναι γενικά γνωστή [1, 6]. Θα μπορούσαμε να βρούμε ικανοποιητικά β και γ δοκιμάζοντας πολλούς συνδυασμούς, αλλά το πρόβλημα γίνεται αρκετά περίπλοκο και η προσέγγιση αυτή δεν μας δίνει κάποια οριστική λύση.

Εδώ (όπως αναφέραμε στην εισαγωγή) εμπνεόμαστε από το quantum adiabatic computing για την επιλογή των β και γ. Αρχικά ο συντελεστής ανάμειξης β είναι πολύ μεγάλος για να είναι πιο γρήγορη η σύγκλιση και ο συντελεστής κέρδους γ είναι πολύ μικρός, γιατί είμαστε αρκετά μακριά από την βέλτιστη λύση. Και όσο οι επαναλήψεις προχωράνε ο συντελεστής ανάμειξης β πάει στο μηδέν και ο συντελεστής κέρδους γ πάει στο 1, γιατί πλησιάζουμε την λύση του προβλήματος.

Έτσι μιμούμενοι μια αδιαβατική συμπεριφορά ορίζουμε για την k-οστή επανάληψη του κυκλώματος [1, 4, 7]:

$$\begin{aligned} \beta_k &= 1 - \frac{k}{p}, \quad \beta \overset{k\to p}{\to} 0 \\ \gamma_k &= \frac{k}{p}, \quad \gamma_k \overset{k\to p}{\to} 1 \end{aligned} \tag{6}$$

2.1 Αριθμός Qubits

Αν είναι n οι επιλογές $|z\rangle = |z_1 z_2 ... z_n\rangle$, τότε θα χρειαστώ οπωσδήποτε n qubits για κάθε μια από αυτές, τα οποία στο τέλος θα μετρήσω. Επίσης θα χρειαστώ ακόμα έξτρα qubits για να αθροίσω τις φθορές της μπαταρίας. Το μέγεθος αυτό είναι το πολύ $\sum_t max(c_1^{(t)}, c_2^{(t)})$, άρα θα χρειαστώ το πολύ $d = \left\lceil \log_2 \left(\sum_t max(c_1^{(t)}, c_2^{(t)}) \right) \right\rceil$ βοηθητικά qubits για να αποθηκεύουν το τρέχον κόστος. Το σύνολο των βοηθητικών αυτών qubits θα το αναπαριστούμε με έναν πίνακα $A[i], 0 \leq i \leq d-1$. Επίσης θα χρειαστώ ένα ακόμα qubit (το Flag qubit) για τον έλεγχο:

$$F = \left[cost(z) = \left(\sum_{t=1}^{n} (1 - z_t) c_1^{(t)} + z_t c_2^{(t)} \leq C_{max} \right) \right]_{0|1}$$

2.2 Αρχιτεκτονική και Χαμιλτονιανή

Η αντικειμενική συνάρτηση που πρέπει να μεγιστοποιήσω είναι η $f(z) = return(z) + penalty(z)$ (ποινή μεγαλύτερης βαρύτητας δεν δίνει επιθυμητά αποτελέσματα [1], οπότε προτιμάμε γραμμική συμπεριφορά):

$$return(z = z_1 z_2 ... z_n) = \sum_{t=1}^{n} \left[(1 - z_t) \lambda_1^{(t)} + z_t \lambda_2^{(t)} \right]$$

$$penalty(z = z_1 z_2 ... z_n) = \left\{ \begin{array}{ll} 0 & \text{εάν } cost(z) \leq C_{max} \\ -a(cost(z) - C_{max}) & \text{εάν } cost(z) > C_{max} \end{array} \right\}$$

$$f(z) = return(z) + penalty(z)$$

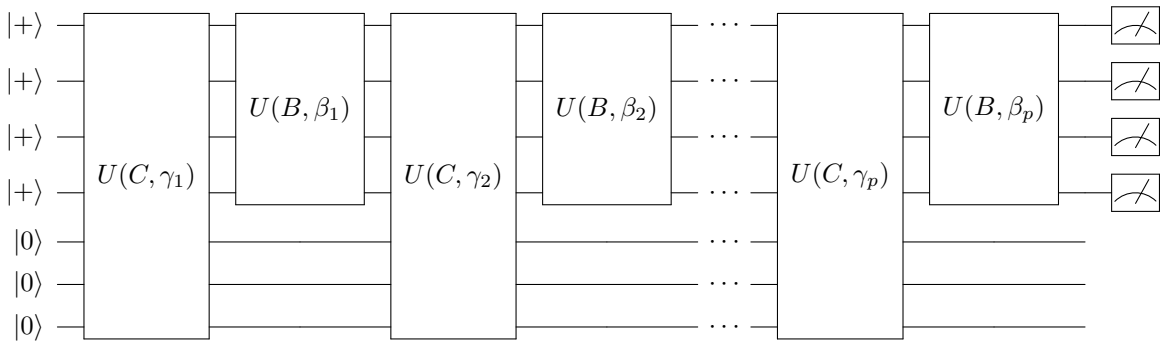
Αρχικά θέτω την κατάσταση του συστήματος σε $|z\rangle = |+\rangle^{\otimes n}$, εφαρμόζοντας μια πύλη Hadamard σε κάθε qubit. Δηλαδή η αρχική κατανομή πιθανότητας των z_i είναι η ομοιόμορφη, με πιθανότητες 50-50 για τις καταστάσεις $|0\rangle$ και $|1\rangle$. Αρχικά λοιπόν έχω $|z\rangle \otimes |\mathbf{0}\rangle \otimes |0\rangle$.

Ύστερα φτιάχνω την $|z\rangle \otimes |cost(z)\rangle \otimes |0\rangle$ και τέλος ενημερώνω το Flag qubit:
$|z\rangle \otimes |cost(z)\rangle \otimes |cost(z) \geq C_{max}\rangle$

Αν $Flag = 1$, τότε επιβάλω ποινή στην λύση μου, η οποία τοποθετείται στην φάση της κατάστασης και συγκεκριμένα στο $|cost(z)\rangle$. Έτσι κατασκευάζω την $|z'\rangle \otimes |cost(z')\rangle \otimes |F'\rangle$.

Μετά, επαναρχικοποιώ τα βοηθητικά qubits και ξαναφτιάχνω την κατάσταση $|z'\rangle \otimes |0\rangle \otimes |0\rangle$. Με αυτή την αντίστροφη διαδικασία τα βοηθητικά qubits επαναρχικοποιούνται σε $|0\rangle$, ενώ η ποινή διατηρείται. Η ιδέα είναι πως αν η αρχική διαδικασία υπολογίζει το $|cost(z)\rangle$ από το $|z\rangle$, τότε η αντίστροφη υπολογίζει το $|z'\rangle$ από το $|cost(z')\rangle$ (και ταυτόχρονα αρχικοποιεί τα βοηθητικά qubits).

Επαναλαμβάνω τα μέρη $U(C, \gamma_k)$ και $U(B, \beta_k)$ p-φορές και στο τέλος μετράω τα n qubits.



Σχήμα 1: QAOA κύκλωμα

Το κύκλωμα αυξάνει γραμμικά με την αύξηση του p και δημιουργεί την κατάσταση:

$$|\beta, \gamma\rangle \equiv U(B, \beta_p)U(C, \gamma_p) \cdots U(B, \beta_1)U(C, \gamma_1) |+\rangle^{\otimes n}$$

2.3 Τελεστής C

Κατασκευάζουμε τον τελεστή C με βάση την αντικειμενική συνάρτηση f:

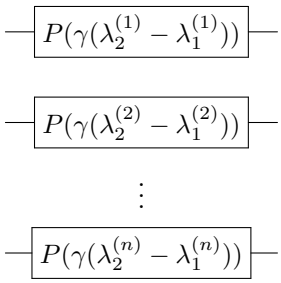
$$\begin{aligned} U(C,\gamma) &= e^{-i\gamma f(z)} |z\rangle \\ &= e^{-i\gamma.penalty(z)} e^{-i\gamma.return(z)} |z\rangle \end{aligned} \tag{7}$$

2.3.1 Υπολογισμός Κερδών

Με βάση την περίπτωση 3 του Battery Revenue Optimization (που αναφέραμε στην εισαγωγή) θεωρούμε πως τα κέρδη θα είναι τουλάχιστον $\theta = \sum_{t=1}^{n} \lambda_1^{(t)}$. Επειδή, το ποσό ϑ είναι ανεξάρτητο της επιλογής z, το αγνοούμε και εκφράζουμε το καθαρό κέρδος ως $(\lambda_2^{(t)} - \lambda_1^{(t)})$, για κάθε t:

$$\begin{aligned} e^{-i\gamma.return(z)} |z\rangle &= \prod_{t=1}^{n} e^{-i\gamma.return_t(z)} |z\rangle \\ &= \cancel{e^{i\theta}} \bigotimes_{t=1}^{n} e^{-i\gamma z_t(\lambda_2^{(t)} - \lambda_1^{(t)})} |z_t\rangle \end{aligned} \tag{8}$$

Υπολογίζουμε παράλληλα τους όρους $e^{-i\gamma(\lambda_2^{(t)} - \lambda_1^{(t)})z_t} |z_t\rangle$ με πίνακες φάσης $P(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix}$:



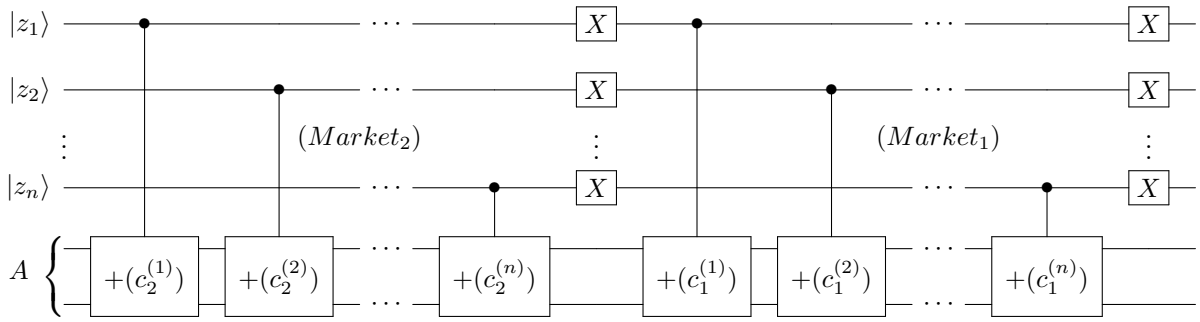
Σχήμα 2: Υπολογισμός κερδών

2.3.2 Υπολογισμός Κόστους

Για κάθε z_i που αντιστοιχεί στην επιλογή του $\text{market}_{1|2}$ την ημέρα i έχω:

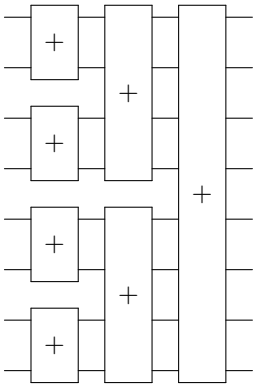
- Για $z_i = 0$ επιλέγω το market M1
- Για $z_i = 1$ επιλέγω το market M2

Και προσθέτω όλες τις τιμές $c_2^{(i)}$ ή $c_1^{(i)}$ υπο τις συνθήκες των z_i στα βοηθητικα qubits $A[i]$:



Σχήμα 3: Υπολογισμός κόστους

Για να μην προσθέτω όλα τα $c^{(i)}$ στην σειρά μπορώ να τα προσθέτω παράλληλα. Με μια τεχνική "διαίρει και βασίλευε": αντί να κάνω n προσθέσεις στην σειρά, κάνω $n/2$ και $n/2$ και μετά προσθέτω τα δύο αποτελέσματα. Αντίστοιχα, αντί να κάνω $n/2$ προσθέσεις στην σειρά, κάνω $n/4$ και $n/4$ κοκ. Έτσι έχω ένα κύκλωμα αθροιστών με βάθος $log_2(n)$. Ωστόσο, η προσέγγιση αυτή απορρίπτεται, επειδή χρησιμοποιεί πολλά περισσότερα qubits (και το κάθε qubit είναι πολύτιμο).



Σχήμα 4: Παράλληλος Υπολογισμός Κόστους

Συγκεκριμένα η σύγκριση των 2 προσεγγίσεων φαίνεται στον παρακάτω πίνακα:

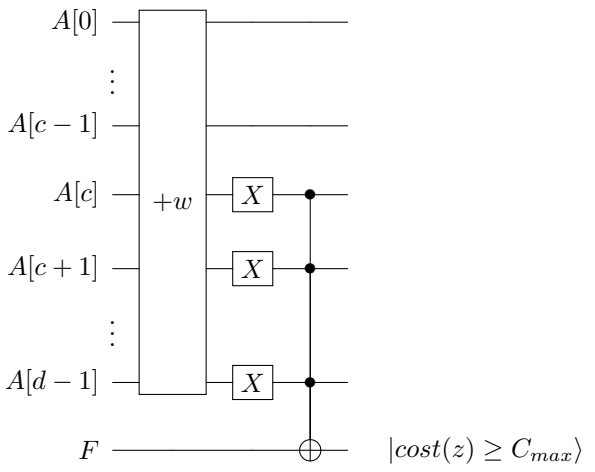
Μέθοδος	Βάθος Κυκλώματος	Βοηθητικά qubits
Σειριακοί Αθροιστές	$O(nlog_2n)$	$O(n)$
Παράλληλες Αθροιστές	$O((log_2n)^3)$	$O(nlog_2(n))$

2.3.3 Έλεγχος περιορισμού

Έχοντας σε δυαδική αναπαράσταση το κόστος της επιλογής z, θέτω το Flag Qubit $= 1$ αν ξεπέρασα το C_{max}. Το C_{max} είναι ένας αυθαίρετος αριθμός, οπότε η σύγκριση είναι εν γένει δύσκολη. Αντιθέτως η σύγκριση με δυνάμεις του 2 είναι πολύ εύκολη. Οπότε προσθέτω μια σταθερή τιμή w στο C_{max}, για να φτάσω στην αμέσως μεγαλύτερη δύναμη του 2. Το ίδιο w προσθέτω και στο κόστος μου:

$$\begin{aligned} cost(z) \leq C_{max} \quad &\overset{+w}{\Longleftrightarrow} \\ cost(z) + w \leq C_{max} + w = 2^c \end{aligned} \tag{9}$$

Τώρα θέτω το flag αν όλα τα qubits πάνω από την δύναμη του 2 είναι μηδέν. Αυτό γίνεται με μια πολλαπλά ελεγχόμενη πύλη NOT, η οποία χρησιμοποιεί $(d - c - 3)$ επικουρικά qubits.



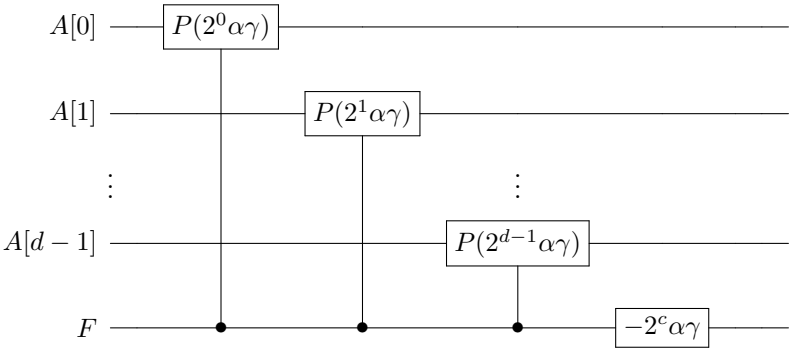
Σχήμα 5: Έλεγχος περιορισμού

2.3.4 Επιβολή ποινής

Αν το $F = 1$, τότε πρέπει να επιβληθεί η ποινή:

$$a(cost(z) - C_{max}) = \sum_{j=0}^{d-1} 2^j aA[j] - 2^c a \tag{10}$$

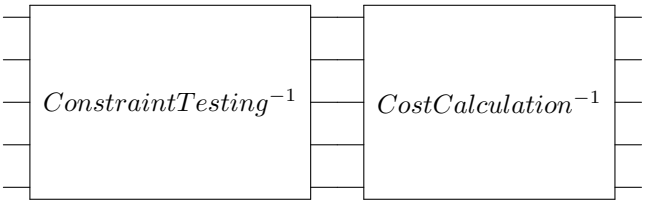
Η ποινή αυτή τοποθετείται στην φάση των qubits. Αν το εκάστοτε qubit είναι μηδέν δεν παθαίνει τίποτα, αν όμως είναι 1 δρα πάνω του η αντίστοιχη φάση:



Σχήμα 6: Επιβολή ποινής

2.3.5 Επαναρχικοποίηση

Για να ξαναμηδενίσω τα βοηθητικά qubits, χωρίς να καταστρέψω την κατάσταση, εφαρμόζω τα κυκλώματα ελέγχου περιορισμού και υπολογισμού κόστους ανάποδα. Με την αντίστροφη πορεία τα qubits θα ξαναπάνε στην κατάσταση $|0\rangle$ και θα διατηρηθεί η φάση από την επιβολή ποινής (αφού ο υπολογισμός κόστους και ο έλεγχος δεν επηρεάζουν την φάση του συστήματος).



Σχήμα 7: Επαναρχικοποίηση

2.4 Τελεστής B

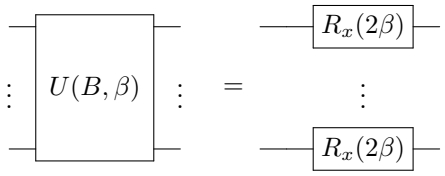
Το δεύτερο κομμάτι είναι και το πιο απλό: Για κάθε qubit εφαρμόζω την πύλη $X = NOT$ με έναν πολλαπλασιαστικό παράγοντα β. Οπότε αν σ_t^x ο τελεστής $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ εφαρμοζόμενος στο t-οστό qubit, ορίζω τον τελεστή B ως το άθροισμα όλων των σ^x:

$$B = \sum_{t=1}^{n} \sigma_t^x \tag{11}$$

και τον αντίστοιχο μονιαδιαίο τελεστή:

$$U(B, \beta) = e^{-i\beta B} \tag{12}$$

Ως κύκλωμα τώρα ο τελεστής $U(B, \beta)$ μπορεί να υλοποιηθεί με πύλες $R_x(2\beta)$ πάνω σε κάθε qubit.



Σχήμα 8: Κύκλωμα Τελεστή Ανάμειξης

3 Βελτιώσεις

3.1 Αναγωγή στο πρόβλημα Relaxed 0-1 Knapsack

Η πρώτη βελτίωση είναι η αναγωγή του Battery Revenue Optimization Problem στο 0-1 Knapsack Problem (όπως αναφέραμε και στην εισαγωγή).

0-1 Knapsack Problem: Έχοντας n αντικείμενα το καθένα με βάρος w_i και αξία v_i, πρέπει να καθορίσω ποιά αντικείμενα θα βάλω σε μια τσάντα η οποία αντέχει μέγιστο βάρος W_{max}, έτσι ώστε να μεγιστοποιήσω την αξία της επιλογής μου.

Τυπικά, λοιπόν, το πρόβλημα έχει τον εξής μαθηματικό φορμαλισμό:

$$\begin{aligned} &\underset{z_t \in \{0,1\}^n}{\operatorname{argmax}} \sum_{t=1}^{n} z_t v_t \\ &\text{τ.ω:} \sum_{t=1}^{n} z_t w_t \leq W_{max} \end{aligned} \tag{13}$$

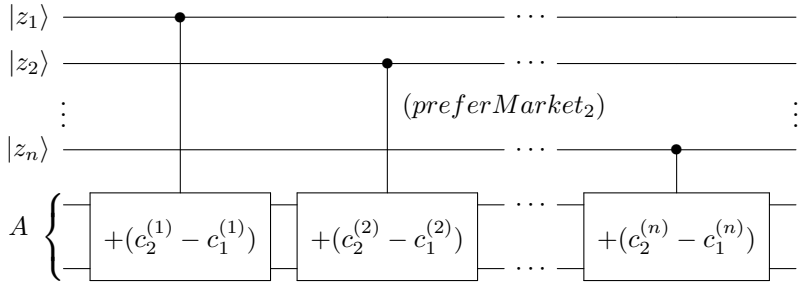
Όπως εξηγήσαμε και παραπάνω, χωρίς βλάβη της γενικότητας, επιλέγουμε την εκδοχή 3 του προβλήματος Battery Revenue Optimization. Εύκολα το πρόβλημα μας ανάγεται στο 0-1 Knapsack αν θέσουμε:

$$\begin{aligned} v_t &= (\lambda_2^{(t)} - \lambda_1^{(t)}) \\ w_t &= (c_2^{(t)} - c_1^{(t)}) \\ W_{max} &= C_{max} - \sum_{t=1}^{n} c_1^{(t)} \end{aligned} \tag{14}$$

Η βασική ιδέα της αναγωγής είναι η εξής: Εφόσον πρέπει αναγκαστικά κάθε μέρα να διαλέξω ένα από τα δύο markets και επειδή το πρώτο (από την εκδοχή 3) έχει μικρότερες τιμές για τα λ_i και c_i, στην ουσία με ενδιαφέρει μόνο για ποιές μέρες προτιμώ το market 2. Οπότε επιλέγω ή δεν επιλέγω το market 2, όπως επιλέγω ή δεν επιλέγω κάθε αντικείμενο που θα τοποθετήσω στην τσάντα μου στο πρόβλημα 0-1 Knapsack.

Με αυτή την τεχνική οι προσθέσεις στο κύκλωμα μειώνονται στις μισές, αφού αθροίζω τις διαφορές $(c_2^{(t)} - c_1^{(t)})$ για κάθε i, και όχι τα $c_1^{(i)}$, $c_2^{(i)}$ ξεχωριστά. Επίσης, χρησιμοποιούσα πύλες X για τον έλεγχο των αθροίσεων, αλλά τώρα δεν χρειάζεται αφού προσθέτω μόνο αν $z_i = 1$.

Η τροποποίηση αυτή μας γλιτώνει n προσθέσεις και $2n$ πύλες X σε κάθε επανάληψη.



Σχήμα 9: Βελτιωμένος Υπολογισμός κόστους

3.2 QFT αθροιστής

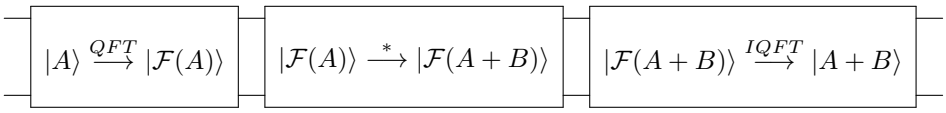
Υπάρχουν πολλά κβαντικά κυκλώματα που υλοποιούν δυαδικό αθροιστή. Μεταξύ αυτών είναι:

- Plain adder network [8]
- Ripple carry adder [9]
- QFT adder [10, 11]

Εμείς θα χρησιμοποιήσουμε τον QFT adder, γιατί έχει ένα σημαντικό πλεονέκτημα στο κύκλωμά μας: έχουμε πολλές αθροίσεις στην σειρά.

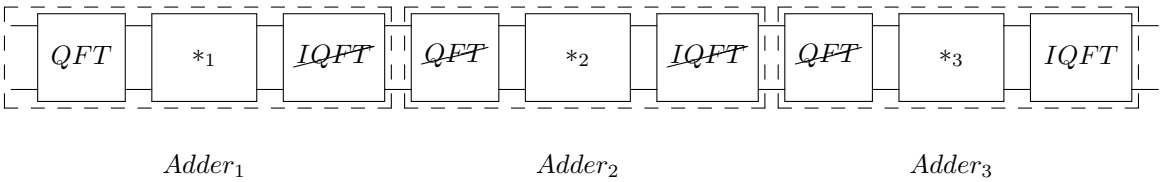
Ο QFT adder προσθέτει τις καταστάσεις $|A\rangle$ και $|B\rangle$, που αντιστοιχούν σε δυαδικές αναπαραστάσεις αριθμών εφαρμόζοντας την πρόσθεση στο πεδίο των φάσεων:

- Εφαρμόζει τον κβαντικό **μετασχηματισμό Fourier** στον αριθμό $|A\rangle \overset{QFT}{\longrightarrow} |\mathcal{F}(\mathcal{A})\rangle$
- Εφαρμόζει **ελεγχόμενες πύλες στροφής** με τα qubits του B ως control qubits και τα qubits του A ως target qubits: $|\mathcal{F}(\mathcal{A})\rangle \overset{B_{phase}}{\longrightarrow} |\mathcal{F}(\mathcal{A}+\mathcal{B})\rangle$
- Εφαρμόζει τον ***αντίστροφο* κβαντικό μετασχηματισμό Fourier** $|\mathcal{F}(\mathcal{A}+\mathcal{B})\rangle \overset{IQFT}{\longrightarrow} |A+B\rangle$



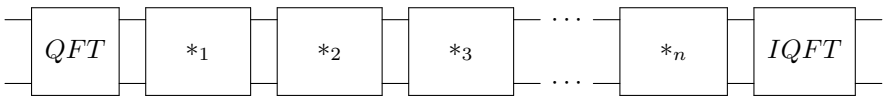
Σχήμα 10: QFT αθροιστής

Δηλαδή η πράξη των περιστροφών στο πεδίο των φάσεων ισοδυναμεί με την κλασική πρόσθεση αριθμών στο δυαδικό σύστημα. Το πλεονέκτημα που δίνει αυτή η προσέγγιση είναι οτι χρειάζεται να κάνουμε **μόνο μια φορά** τους μετασχηματισμούς QFT και IQFT. Οπότε το:



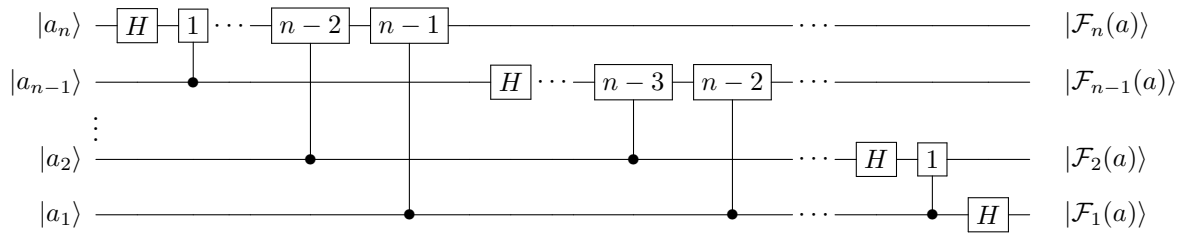
Σχήμα 11: Σειριακός αθροιστής

μετασχηματίζεται στο:



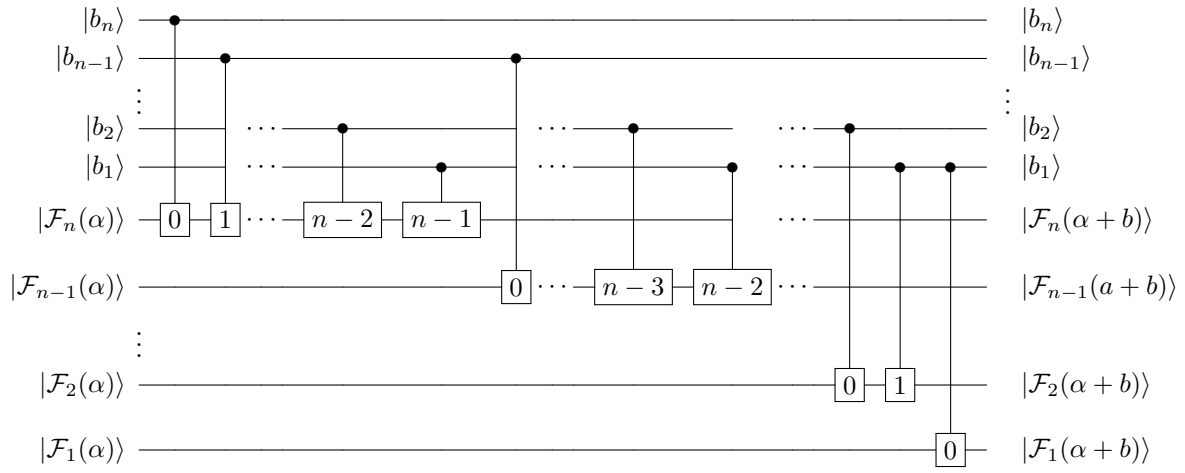
Σχήμα 12: Βελτιωμένος σειριακός αθροιστής

Τα δύο κυκλώματα κατασκευάζονται ως εξής:



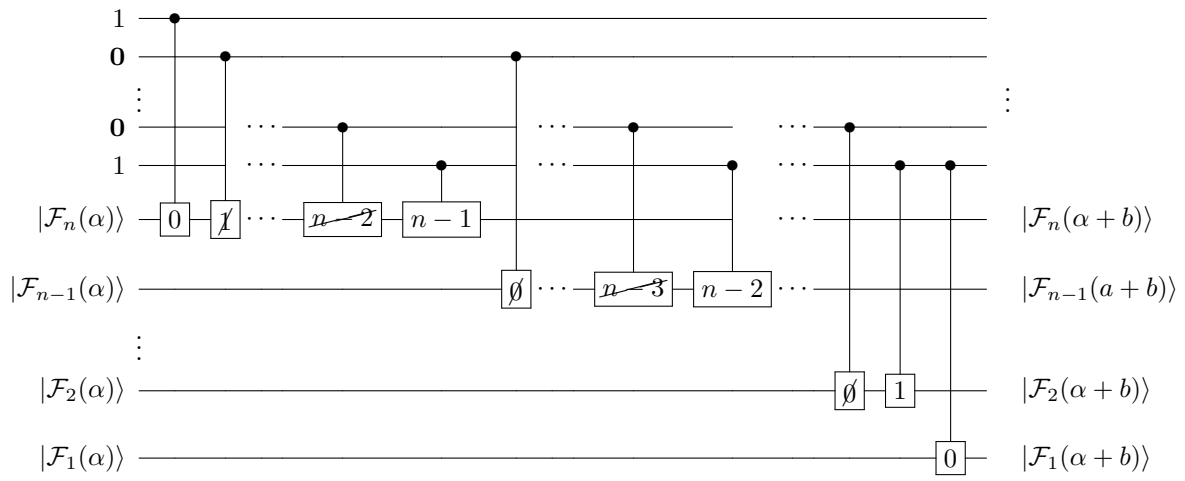
Σχήμα 13: QFT κύκλωμα

όπου $\boxed{k}$ η πύλη που αντιστοιχεί στον πίνακα φάσης $P(\frac{\pi}{2^k}) = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{2^k}} \end{pmatrix}$.



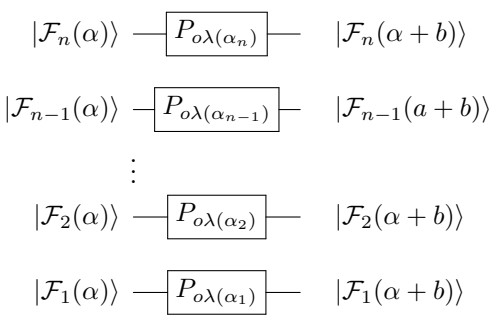
Σχήμα 14: Υπορουτίνα περιστροφών

Ένα ακόμα πλεονέκτημα είναι πως στην περίπτωσή μας ο αριθμός $|B\rangle$ είναι γνωστός, άρα εκφράσιμος με κλάσικά bits. Οπότε οι ελεγχόμενες πύλες στροφής για τα qubits του A μπορούν να αντικατασταθούν από απλές πύλες στροφής, όταν το αντίστοιχο b_i είναι ίσο με 1 και καθόλου πύλες όταν το αντίστοιχο b_i είναι ίσο με 0.



Σχήμα 15: Υπορουτίνα περιστροφών χωρίς control gates

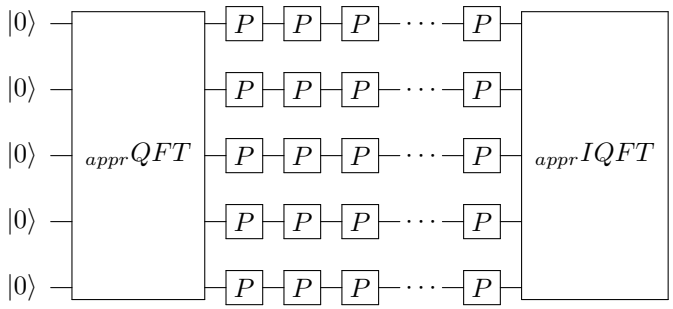
Για τις απλές πύλες στροφής όμως ισχύει: $P(\varphi) * P(\psi) = P(\varphi + \psi)$, οπότε όλες οι πύλες που αντιστοιχούν σε ένα qubit a_i ενώνονται σε μία.



Σχήμα 16: Βελτιωμένη υπορουτίνα περιστροφών

Επίσης για τον μετασχηματισμό QFT μπορούμε να αγνοήσουμε γωνίες αρκετά μικρές μιας και ο θόρυβος στα πραγματικά συστήματα είναι συγκρίσιμος της πράξης. Συγκεκριμένα το μέγιστο k που μας αρκεί είναι $k \approx log_2(n)$. Έτσι ο αθροιστής, από $\frac{1}{2}n(n+1) = O(n^2)$ πύλες, καταλήγει να έχει $O(nlog_2 n)$ πύλες [10]. Οπότε χρησιμοποιούμε τον Approximate QFT [12].

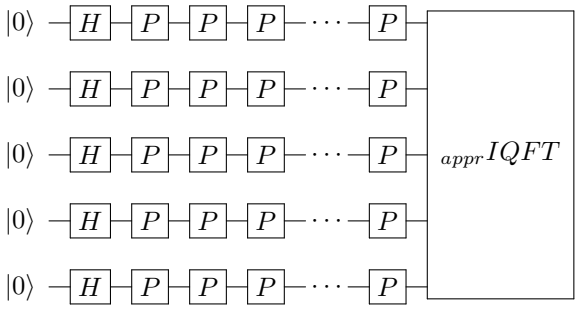
Με την επιλογή του αθροιστή ${}_{appr}QFT$ και τις βελτιώσεις μας, έχουμε εν τέλει ένα πλέγμα με πύλες φάσεων για τον υπολογισμό του κόστους:



Σχήμα 17: Βελτιωμένο κύκλωμα υπολογισμού κόστους

(Σημείωση: Επειδή και στον έλεγχο του περιορισμού έχουμε μια άθροιση μπορούμε να καθυστερήσουμε το IQFT προσθέτοντας την σταθερή τιμή w εκεί).

Ωστόσο, παρατηρούμε πως σε κάθε επανάληψη του κυκλώματος το αρχικό QFT δρά πάνω σε qubits που βρίσκονται στην κατάσταση $|0\rangle$. Οπότε, οι πύλες στροφής δεν έχουν νόημα και το QFT εκφυλίζεται σε απλές πύλες Hadamard. Άρα, έχουμε εν τέλει:



Σχήμα 18: Τελικό κύκλωμα υπολογισμού κόστους

3.3 Αποφυγή Flag Qubit

Στο κύκλωμα που ελέγχει το κόστος χρησιμοποιούσαμε το επιπλέον Flag Qubit. Προσθέταμε την τιμή w που λείπει από το C_{max}, ώστε να φτάσουμε στην αμέσως επόμενη δύναμη του 2 (2^c) και μετά κάναμε έλεγχο με την πολλαπλά ελεγχόμενη πύλη NOT για το Flag (η οποία χρησιμοποιεί $d-c-3$ βοηθητικά qubits).

Εδώ προτείνουμε το εξής: Αντί να φτάσουμε στην αμέσως επόμενη δύναμη $2^c \geq C_{max}$, στοχεύουμε στην μέγιστη δύναμη του 2, που χωράει στα βοηθητικά μας qubits.

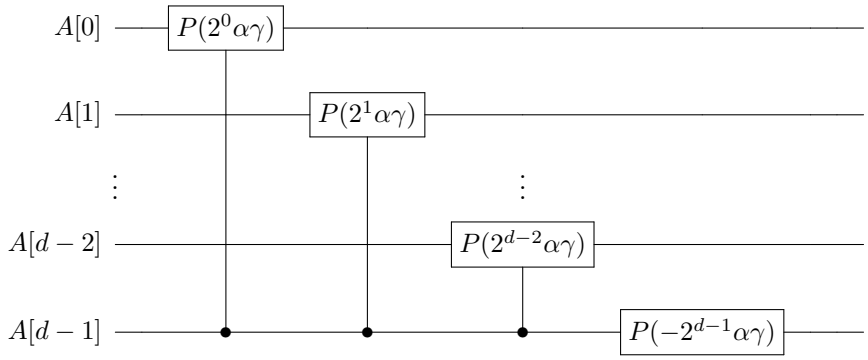
Δηλαδή: $w' = 2^d - C_{max}$.

Η ιδέα είναι η ίδια: Προσθέτω το νέο w' και στα δύο μέλη της ανίσωσης: $cost(z) < C_{max} \Rightarrow cost(z) + w' < C_{max} + w' = 2^d$, οπότε το Flag είναι το τελευταίο μας Qubit και έτσι αποφεύγουμε την Multi-CNOT.

Προσθέτουμε λοιπόν w' στα δεδομένα μας:

- Αν το τελευταίο Qubit d έγινε 1, τότε $cost(z) > Cmax$
- Αν το τελευταίο Qubit d παρέμεινε 0, τότε $cost(z) \leq Cmax$

Γλιτώνουμε $d-c$ πύλες X και $d-c-3$ βοηθητικά qubits. Φυσικά τροποποιούμε και το κύκλωμα επιβολής ποινής: Οι πύλες ελέγχονται από το τελευταίο Qubit μας και όχι από το Flag.



Σχήμα 19: Βελτιωμένο κύκλωμα επιβολής ποινής

3.4 Αύξηση ακρίβειας

Στο κύκλωμά μας εφαρμόζουμε αρχικά την πύλη Hadamard για κάθε Qubit, δηλαδή ορίζουμε μια ομοιόμορφη κατανομή με πιθανότητες 50-50. Αυτή η επιλογή, ωστόσο, είναι εντελώς αυθαίρετη και θα έπρεπε να επιλέξουμε μια καλύτερη.

Φυσικά για άλλη κατανομή θα χρειαστούμε άλλον τελεστή ανάμειξης, έναν που να ανταποκρίνεται σε αυτήν. Για την αρχική κατανομή $|+\rangle^{\otimes n}$ χρειαστήκαμε πύλη X, επειδή αυτή έχει τις ιδιοκαταστάσεις $(|0\rangle + |1\rangle)/\sqrt{2}$ και $(|0\rangle - |1\rangle)/\sqrt{2}$. Όμως θέλω έναν μείκτη για την τυχούσα κατάσταση [6]:

$$
\begin{aligned}
|p_i\rangle &\coloneqq \sqrt{1-p_i}\,|0\rangle + \sqrt{p_i}\,|1\rangle \\
|p_i^{\perp}\rangle &\coloneqq -\sqrt{p_i}\,|0\rangle + \sqrt{1-p_i}\,|1\rangle \\
|p\rangle &= |p_1\rangle \otimes |p_2\rangle \otimes \cdots |p_n\rangle\,.
\end{aligned}
\tag{15}
$$

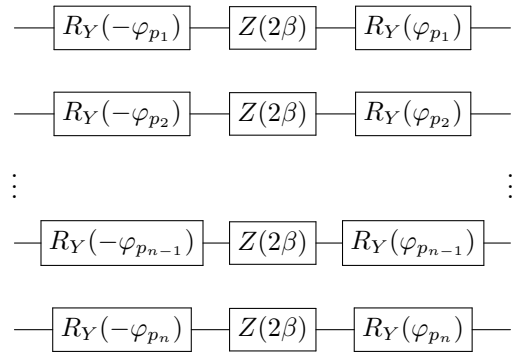
Χρειάζομαι λοιπόν έναν μείκτη $\mathbf{X}$ με ιδιοκαταστάσεις: $\mathbf{X}_{p_i}\,|p_i\rangle = -\,|p_i\rangle$ και $\mathbf{X}_{p_i}\,|p_i^{\perp}\rangle = +\,|p_i\rangle$. Παρατηρούμε πως η πύλη Z έχει τις ιδιοκαταστάσεις $|0\rangle$ και $|1\rangle$, οπότε ο $\mathbf{X}$ ορίζεται ως γραμμικός συνδυασμός των X και Z (γι'αυτό και το σύμβολο $\mathbf{X}$, ως υπέρθεσή των):

$$
\begin{aligned}
\mathbf{X}_{p_i} &= -(1-2p_i)Z - 2\sqrt{p_i(1-p_i)}X \\
&= -\begin{pmatrix} 1-2p_i & 2\sqrt{p_i(1-p_i)} \\ 2\sqrt{p_i(1-p_i)} & 1-2p_i \end{pmatrix}
\end{aligned}
\tag{16}
$$

Και φυσικά στις ακραίες περιπτώσεις: $\mathbf{X}_0 = -Z$, $\mathbf{X}_{1/2} = -X$. Το μόνο που μένει είναι να κατασκευαστεί η πύλη αυτή (και προφανώς θα χρησιμοποιήσω πύλες στροφής). Ορίζω τις γωνίες που αντιστοιχούν στις πιθανότητες: $\varphi_{p_i} = 2\sin^{-1}(\sqrt{p_i})$ και ο πίνακας παίρνει την μορφή:

$$
\begin{aligned}
\mathbf{X}_{p_i} &= -\begin{pmatrix} \cos(\varphi_{p_i}) & \sin(\varphi_{p_i}) \\ \\ \sin(\varphi_{p_i}) & -\cos(\varphi_{p_i}) \end{pmatrix} \\
&= -\begin{pmatrix} 1-2\sin^2(\frac{\varphi_{p_i}}{2}) & 2\cos(\frac{\varphi_{p_i}}{2})\sin(\frac{\varphi_{p_i}}{2}) \\ \\ 2\cos(\frac{\varphi_{p_i}}{2})\sin(\frac{\varphi_{p_i}}{2}) & -(1-2\sin^2(\frac{\varphi_{p_i}}{2})) \end{pmatrix}
\end{aligned}
\tag{17}
$$

Οπότε μπορώ να την κατασκευάσω ως εξής: $\mathbf{X}_{p_i} = R_Y(\varphi_{p_i})ZR_Y(\varphi_{p_i})^{\dagger}$ και ο αντίστοιχος μείκτης είναι $e^{-i\beta\mathbf{X}_{p_i}} = R_Y(\varphi_{p_i})e^{-i\beta Z}R_Y(\varphi_{p_i})^{\dagger}$. Για τον πίνακα R_Y ισχύει: $R_Y(\varphi_{p_i})^{\dagger} = R_Y(-\varphi_{p_i})$. Άρα, έχω το παρακάτω κύκλωμα:



Σχήμα 20: Μείκτης για τυχούσα κατανομή p_i

Όσον αφορά την κατανομή: μια αρχική σκέψη είναι να επιβάλουμε μια κατανομή η οποία έχει αναμενόμενα κέρδη ακριβώς C_{max}. Για να το πετύχουμε αυτό θέτουμε την πιθανότητα όλων των Qubits ίση με (Constant Biased State):

$$
\mathbf{Pr}([Q_i = 1]) = \frac{C_{max}}{\sum\limits_t c_i}
\tag{18}
$$

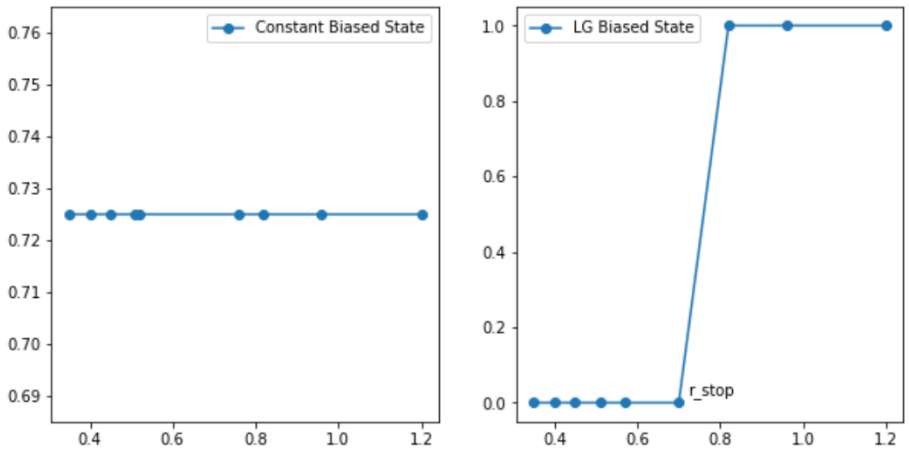
Οπότε στην μέση περίπτωση εκμεταλλευόμαστε το μέγιστο κόστος C_{max}:

$$\mathbf{E}[cost(z)] = \sum_{t=1}^{n} c_i \cdot p_i = \frac{\sum_{t=1}^{n} c_i \cdot C_{max}}{\sum_{t=1}^{n} c_i} = C_{max} \tag{19}$$

Αυτή η προσέγγιση δίνει μια σταθερή κατανομή και δεν λαμβάνει υπόψιν τα δεδομένα ξεχωριστά, παρά στο σύνολό τους. Μια άλλη προσέγγιση είναι να μιμηθώ έναν κλασικό άπληστο αλγόριθμο γνωστό και ως **Lazy Greedy**.

Σύμφωνα με αυτόν ταξινομούμε τα αντικείμενα σε φθίνουσα σειρά με κριτήριο τον λόγο $r_i = \frac{\lambda_i}{c_i}$ και επιλέγουμε τα πρώτα m αντικείμενα μέχρις ότου να ξεπεράσουμε το C_{max}. Είναι γραμμικός σε χρόνο, αλλά σπάνια πετυχαίνει την βέλτιση λύση. Εμπνεόμενοι από τον αλγόριθμο μπορούμε να δημιουργήσουμε μιαν κατανομή που θέτει $p_i = 1$ για τα πρώτα m αντικείμενα και $p_i = 0$ για τα επόμενα (Ορίζουμε r_{stop} τον λόγο r_m του αντικειμένου m και οποιοδήποτε αντικείμενο έχει λόγο μικρότερο από r_{stop} δεν το επιλέγουμε).

Οι δύο κατανομές έχουν αντιδιαμετρικά αντίθετες συμπεριφορές. Η μία δίνει μια ομοιόμορφη κατανομή και η άλλη μια εντελώς ανομοιόμορφη, όπως φαίνονται και στο σχήματα παρακάτω.



Σχήμα 21: Δύο ακραίες κατανομές πιθανοτήτων **Constant** και **Lazy Greedy**

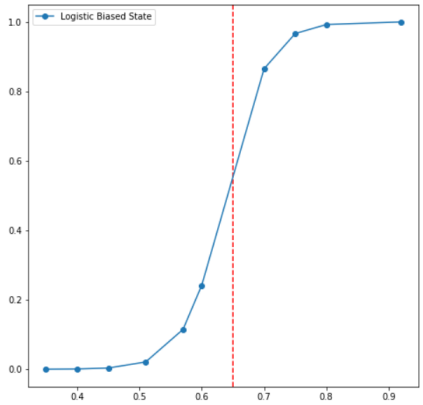
Ας κατασκευάσουμε λοιπόν μια κατανομή η οποία βρίσκεται κάπου μεταξύ των δύο(**Logistic** [6]). Μια παράμετρος k θα ρυθμίζει το κατά πόσο η κατανομή εκφυλίζεται στην Lazy Greedy ή στην Constant Biased:

$$p_i = \frac{1}{1 + Ce^{-k(r_i - r_{stop})}} \tag{20}$$

$$\text{με } C = \frac{\sum_i c_i}{C_{max}} - 1$$

- Με $\lim_{k \to 0} p_i =$ Constant Biased State
- Με $\lim_{k \to \infty} p_i =$ Lazy Greedy State

Η κατανομή αυτή μοιάζει κάπως έτσι (με συμμετρία ως προς r_{stop}):



Σχήμα 22: Ενδιάμεση κατανομή για μικρό k

Με λίγες δοκιμές για το k πετυχαίνουμε μια καλή ακρίβεια.

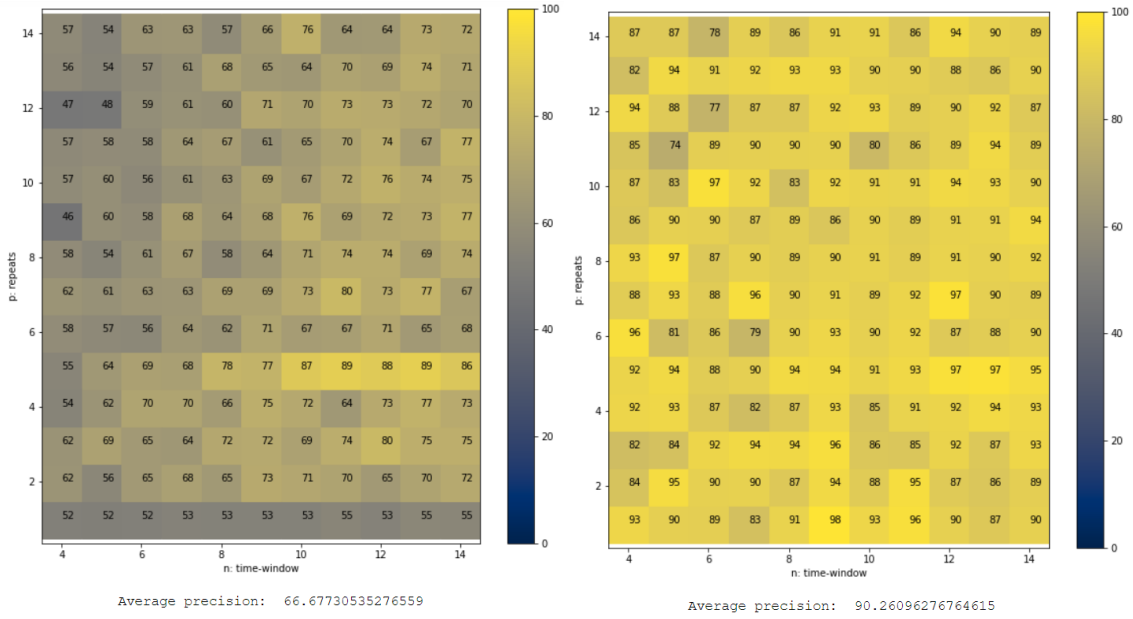
4 Αποτελέσματα

4.1 Μέτρηση ακρίβειας

Εφόσον ο αλγόριθμός μας είναι προσεγγιστικός, πρέπει να μετράμε την ακρίβεια των αποτελεσμάτων μας. Επειδή ο κβαντικός υπολογιστής δίνει διαφορετική απάντηση κάθε φορά χρειάζεται να μετράμε σε τί ποσοστό δίνει αποδεκτές λύσεις ($cost(z) \leq C_{max}$) και πόσο κοντά είναι αυτές οι λύσεις στην βέλτιστη τιμή λ_{opt}. Οπότε ως ακρίβεια μετράμε τα μέσα κέρδη προς τα βέλτιστα κέρδη ($R(z) = return(z) =$ κέρδη της επιλογής z):

$$\frac{\sum\limits_{z}\left[\left(R(z)-\sum\limits_{t}\lambda_1^{(t)}\right)H(cost(z)\leq C_{max})\right]}{N_{feasible}\cdot\left(\lambda_{opt}-\sum\limits_{t}\lambda_1^{(t)}\right)} \tag{21}$$

Η ακρίβεια για την $|+\rangle^{\otimes n}$ και την Logistic κατανομή (με $k=5$) φαίνεται στο παρακάτω συγκριτικό σχήμα:



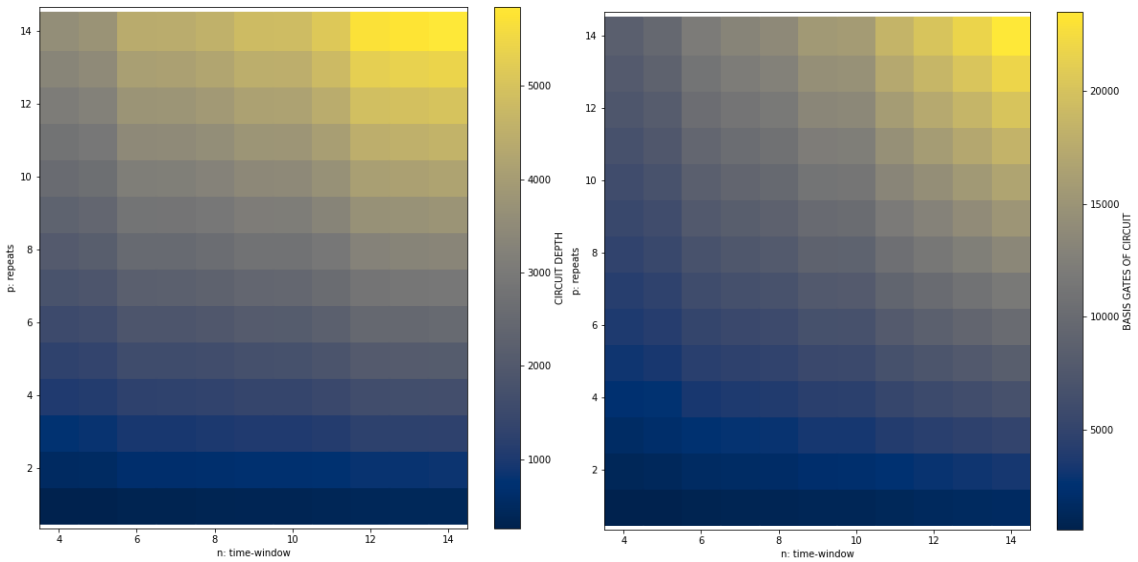
Σχήμα 23: Ακρίβεια κυκλώματος με κατανομες $|+\rangle^{\otimes n}$ και **Logistic** ($k=5$) [επί 100%]

4.2 Κόστος κυκλώματος

Για το κόστος του κυκλώματος μας ενδιαφέρει κυρίως το βάθος του (αφού αυτό καθορίζει τον χρόνο υπολογισμού) και το πλήθος των πυλών που χρησιμοποιεί.

Μιας και μιλάμε για προσομοιωτή της IBM θα αναλύσουμε το κύκλωμα στις βασικές πύλες rz,sx,cx με την εντολή transpile(QCircuit, basis_gates=['cx', 'rz','sx']).

Τα αποτελέσματα των προσομοιώσεων για διάφορα ζεύγη (n, p) φαίνονται στους παρακάτω πίνακες (για το βελτιωμένο κύκλωμα φυσικά):



Σχήμα 24: Βάθος και πλήθος πυλών κυκλώματος

Όπως φαίνεται και από τα σχήματα το κόστος και ο χρόνος του κυκλώματος αυξάνουν γραμμικά σε σχέση με το μήκος της εισόδου n και τις επαναλήψεις του αλγορίθμου p, όπως ήταν αναμενόμενο.

5 Επίλογος

5.1 Συμπεράσματα

Όπως φάνηκε από την εργασία αυτή, μια λύση του Battery Revenue Optimization Problem είναι εφικτή σε κβαντικό υπολογιστή με την προσέγγιση QAOA. Με την κατανομή Logistic πετύχαμε ικανοποιητική ακρίβεια 0.9. Παρατηρούμε, ωστόσο, οτι η ακρίβεια δεν βελτιώνεται για μεγαλύτερο p: αυτό ίσως οφείλεται στις παραμέτρους (β_k, γ_k), οι οποίες δεν συνεισφέρουν στον αλγόριθμο σημαντικά όταν $k \to p$, οπότε το αποτέλεσμα επηρεάζεται από τυχαίες παραμέτρους όπως ο θόρυβος. Παρόλαυτά, για σχετικά μικρό p η ακρίβεια είναι αρκετή και ο χρόνος εκτέλεσης του αλγορίθμου πολύ μικρός.

5.2 Σκέψεις για περαιτέρω βελτίωση

Για να βελτιώσουμε περαιτέρω το κύκλωμα θα μπορούσαμε να δοκιμάσουμε τα εξής:

- Καλύτερη **κατανομή πιθανοτήτων** στα αρχικά qubits.
- Καλύτερους **μείκτες**, κατά προτίμηση πολλών qubits [6], που να λαμβάνουν υπόψιν και την από κοινού συνάρτηση πυκνότητας πιθανότητας: όταν αυξάνονται οι πιθανότητες να είναι άσσος το t-οστό qubit, τότε μειώνονται οι πιθανότητες για όλα τα υπόλοιπα, αφού $\sum_{t=1}^{n} max(c_1^{(t)}, c_2^{(t)}) > C_{max}$. Η προσέγγιση αυτή, ωστόσο, ανεβάζει πολύ το κόστος του κυκλώματος, γιατί αν κάθε qubit επηρεάζεται από όλα τα υπόλοιπα, τότε οι μείκτες αναμένεται να έχουν κόστος $O(n^2)$.
- **Παράλληλη** άθροιση των c_i [1] την οποία αποφύγαμε, γιατί προτιμήσαμε το κύκλωμα με τον ελάχιστο αριθμό qubits.
- Καλύτερους **αθροιστές**.

6 Κώδικας

Αρχικά καλούμε τις απαραίτητες βιβλιοθήκες:

```
from qiskit import QuantumCircuit, QuantumRegister
from typing import List, Union
import math
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, assemble
from qiskit.compiler import transpile
from qiskit.circuit import Gate
from qiskit.circuit.library.standard_gates import *
from qiskit.circuit.library import QFT
```

6.1 (Τελικό) κύκλωμα

```
# knapSack reduced
def solver_function(L: list, C: list, C_max: int, p: int) -> QuantumCircuit:

    # the number of qubits representing answers
    index_qubits = len(L)


    # data qubits and QFT approximation degree
    max_c = sum(C)
    data_qubits = math.ceil(math.log(max_c, 2)) + 1 if not max_c & (max_c - 1) == 0
        else math.ceil(math.log(max_c, 2)) + 2
    appr = math.floor(math.log(data_qubits, 2))


    # LOGISTIC BIASED STATE
    # sort indexes by ratio in ratios zipped list
    ratios = zip([l/c for (l,c) in zip(L,C)], range(n))
    ratios = sorted(ratios, key = lambda t: t[0], reverse=True)

    # find r_stop
    r_sum , r_stop = 0, 0
    for i in range(index_qubits):
        r_stop = ratios[i][0]
        if r_sum + C[i] > C_max:
            break
        else:
            r_sum += C[i]

    # find phi_i for every p_i
    phi = [0]*index_qubits
    for i in range(index_qubits):
        # parameters
        C_const = sum(C)/C_max - 1
        r_i = L[i] / C[i]
        k = 5

        # p_i and phi_i
        p_i = 1 / ( 1 + C_const * math.e**(-k*(r_i - r_stop)) )
        phi[i] = 2 * math.asin( math.sqrt(p_i) )


    def phase_return(index_qubits: int, gamma: float, L: list, to_gate=True)
        -> Union[Gate, QuantumCircuit]:

        qr_index = QuantumRegister(index_qubits, "index")
        qc = QuantumCircuit(qr_index)

        for i in range(index_qubits):
            qc.p( - gamma * L[i] / 2 , qr_index[i])

        return qc.to_gate(label=" phase return ") if to_gate else qc



    def subroutine_add_const(data_qubits: int, const: int, to_gate=True)
        -> Union[Gate, QuantumCircuit]:
```

```
    qc = QuantumCircuit(data_qubits)

    # accumulate phases per qubit
    phases = [0.0] * data_qubits
    for i in range(data_qubits):
        for j in range(i+1):
            if (const>>j)&1 == 1:
                phases[i] += math.pi / (1<<(i-j))

    for i in range(data_qubits):
        qc.p(phases[i], data_qubits-1-i)

    return qc.to_gate(label=" [+"+str(const)+"] ") if to_gate else qc



def cost_calculation(index_qubits: int, data_qubits: int, C: list, to_gate = True)
    -> Union[Gate, QuantumCircuit]:

    qr_index = QuantumRegister(index_qubits, "index")
    qr_data = QuantumRegister(data_qubits, "data")
    qc = QuantumCircuit(qr_index, qr_data)

    # QFT Once here (+ add + add w) then IQFT
    qc = qc.compose(QFT(data_qubits, appr), qr_data[:])

    for i in range(index_qubits):
        gate = subroutine_add_const(data_qubits, C[i]).control()
        qc.append(gate,  [qr_index[i]] + qr_data[:])

    return qc.to_gate(label=" Cost Calculation ") if to_gate else qc



def constraint_testing(data_qubits: int, C_max: int, to_gate = True)
    -> Union[Gate, QuantumCircuit]:

    qr_data = QuantumRegister(data_qubits, "data")
    qc = QuantumCircuit(qr_data)

    # add w'
    rest = 2**(data_qubits-1) - C_max
    qc.append(subroutine_add_const(data_qubits, rest), qr_data[:])
    # IQFT
    qc = qc.compose(QFT(data_qubits, approximation_degree=appr, inverse=True), qr_data[:])


    return qc.to_gate(label=" Constraint Testing ") if to_gate else qc



def penalty_dephasing(data_qubits: int, alpha: float, gamma: float, to_gate = True)
    -> Union[Gate, QuantumCircuit]:

    qr_data = QuantumRegister(data_qubits, "data")
    qc = QuantumCircuit(qr_data)

    # dephase qubits
    for i in range(data_qubits-1):
        qc.cp((2**i)*alpha*gamma, qr_data[data_qubits-1], qr_data[i])

    # dephase yourself
    phase = 2**(data_qubits-1)
    qc.p(phase*alpha*gamma, qr_data[data_qubits-1])

    return qc.to_gate(label=" Penalty Dephasing ") if to_gate else qc



def reinitialization(index_qubits: int, data_qubits: int, C: list, C_max: int, to_gate = True)
    -> Union[Gate, QuantumCircuit]:

    qr_index = QuantumRegister(index_qubits, "index")
    qr_data = QuantumRegister(data_qubits, "data")
```

```
        qc = QuantumCircuit(qr_index, qr_data)

        # clear ancillary qubits
        qc.append(constraint_testing(data_qubits, C_max).inverse(), qr_data[:])
        qc.append(cost_calculation(index_qubits, data_qubits, C).inverse(), qr_index[:] + qr_data[:])

        return qc.to_gate(label=" Reinitialization ") if to_gate else qc



    def mixing_operator(index_qubits: int, beta: float, to_gate = True)
        -> Union[Gate, QuantumCircuit]:

        qr_index = QuantumRegister(index_qubits, "index")
        qc = QuantumCircuit(qr_index)


        # Biased state mixer Ry(theta) exp(-ibetaZ) Ry(-theta)
        # Ry(w)^dagger = Ry(-w)
        for i in range(index_qubits):
            qc.ry(-phi[i], qr_index[i])
            qc.rz(2*beta, qr_index[i])
            qc.ry( phi[i], qr_index[i])



        return qc.to_gate(label=" Mixing Operator ") if to_gate else qc


        ##############################


    qr_index = QuantumRegister(index_qubits, "index")
    qr_data = QuantumRegister(data_qubits, "data")
    cr_index = ClassicalRegister(index_qubits, "c_index")
    qc = QuantumCircuit(qr_index, qr_data, cr_index)


    alpha = 1 # penalty parameter

    # Logistic initial state
    for i in range(index_qubits):
        qc.ry(phi[i], qr_index[i])



    for i in range(p):

        beta  = 1 - (i + 1) / p
        gamma = (i + 1) / p


        qc.append(phase_return(index_qubits, gamma, L), qr_index)
        qc.append(cost_calculation(index_qubits, data_qubits, C), qr_index[:] + qr_data[:])
        qc.append(constraint_testing(data_qubits, C_max), qr_data[:])
        qc.append(penalty_dephasing(data_qubits, alpha, gamma), qr_data[:])
        qc.append(reinitialization(index_qubits, data_qubits, C, C_max), qr_index[:] + qr_data[:])
        qc.append(mixing_operator(index_qubits, beta), qr_index)

    qc.measure(qr_index, cr_index[::-1])

    return qc
```

6.2 Βοηθητικός κλασικός αλγόριθμος με τεχνική DP

Για την μέτρηση της ακρίβειας χρειαζόμαστε έναν κλασικό αλγόριθμο για την λύση του 0-1 KnapSack.

```
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    # Build table K[][] in bottom up manner
```

```
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]
```

6.3 Μέτρηση ακρίβειας και κόστους

```
from qiskit import *
import random
import matplotlib.pyplot as plt
import numpy as np


dataX, dataY = [], []
precisions, depths , gates = [], [], []

for n in range(4, 15):
    for p in range(1,15):

        test_cases = 5
        precision = 0
        qgates = 0
        qdepth = 0

        for t in range(test_cases):

            # Random input of length n
            C = [random.randint(1, 3) for i in range(n)]
            L = [random.randint(1, 4) for i in range(n)]
            C_max = random.randint(2, sum(C)-1)

            # solve classicaly with DP
            l_opt = knapSack(C_max, C, L, n)

            # solve in QC
            qc = solver_function(L, C, C_max, p)
            Ns = 1024//test_cases
            counts = execute(qc, Aer.get_backend('qasm_simulator'), shots=Ns).result().get_counts()
            data = sorted(counts.items(), key = lambda x:x[1], reverse=True)

            # evaluate cost or value of your choice
            def evaluate(L: List[int], a: str):
                s = 0
                for i in range(len(L)):
                    if a[i] == '1':
                        s += L[i]
                return s

            Nf , my_sum = 0, 0
            for i in data:
                cost = evaluate(C, i[0])
                val  = evaluate(L, i[0])
                shots = i[1]
                if cost <= C_max:
                    Nf += shots
                    my_sum += val * shots

            # transpile into basis gates
            qc = transpile(qc, basis_gates=['cx', 'rz','sx'])

            # take measurements
            qdepth += qc.depth()
            qgates += len(qc.get_instructions('cx')) + len(qc.get_instructions('rz')) + len(qc.get_instructions('sx'))
            precision += 100 * my_sum / (Nf * l_opt)

        # take avg of measurements
```

```
        dataX.append(n)
        dataY.append(p)

        precisions.append(precision/test_cases)
        gates.append(qgates/test_cases)
        depths.append(qdepth/test_cases)



norm = mpl.colors.Normalize(vmin=0, vmax=100)
plt.scatter(dataX, dataY, s=1600, c=precisions, cmap=plt.cm.get_cmap("cividis"), norm=norm, marker='s')
plt.xlabel("n: time-window")
plt.ylabel("p: repeats")
for i, z in enumerate(precisions):
    plt.annotate(str(int(z)), (dataX[i], dataY[i]))
plt.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap=plt.cm.get_cmap("cividis")))
plt.show()
print("Average precision: ", sum(precisions) / len(precisions))



plt.scatter(dataX, dataY, s=1600, c=gates, cmap=plt.cm.get_cmap("cividis"), marker='s')
plt.xlabel("n: time-window")
plt.ylabel("p: repeats")
plt.colorbar()
plt.show()



norm = mpl.colors.Normalize(vmin=0, vmax=100)
plt.scatter(dataX, dataY, s=1600, c=depths, cmap=plt.cm.get_cmap("cividis"), norm=norm, marker='s')
plt.xlabel("n: time-window")
plt.ylabel("p: repeats")
plt.colorbar()
plt.show()
```

Αναφορές

[1] Pierre Dupuy de la Grand'rive and Jean-Francois Hullo. "Knapsack problem variants of qaoa for battery revenue optimisation". In: *arXiv preprint arXiv:1908.02210* (2019).

[2] Ce Jin. "An improved FPTAS for 0-1 knapsack". In: *arXiv preprint arXiv:1904.09562* (2019).

[3] Lauren Pusey-Nazzaro et al. "Adiabatic quantum optimization fails to solve the knapsack problem". In: *arXiv preprint arXiv:2008.07456* (2020).

[4] Mark W Coffey. "Adiabatic quantum computing solution of the knapsack problem". In: *arXiv preprint arXiv:1701.05584* (2017).

[5] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A quantum approximate optimization algorithm". In: *arXiv preprint arXiv:1411.4028* (2014).

[6] Wim van Dam et al. "Quantum Optimization Heuristics with an Application to Knapsack Problems". In: *arXiv preprint arXiv:2108.08805* (2021).

[7] Stefan H Sack and Maksym Serbyn. "Quantum annealing initialization of the quantum approximate optimization algorithm". In: *arXiv preprint arXiv:2101.05742* (2021).

[8] Vlatko Vedral, Adriano Barenco, and Artur Ekert. "Quantum networks for elementary arithmetic operations". In: *Physical Review A* 54.1 (1996), p. 147.

[9] Steven A Cuccaro et al. "A new quantum ripple-carry addition circuit". In: *arXiv preprint quant-ph/0410184* (2004).

[10] Thomas G Draper. "Addition on a quantum computer". In: *arXiv preprint quant-ph/0008033* (2000).

[11] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. "Quantum arithmetic with the quantum Fourier transform". In: *Quantum Information Processing* 16.6 (2017), p. 152.

[12] Adriano Barenco et al. "Approximate quantum Fourier transform and decoherence". In: *Physical Review A* 54.1 (1996), p. 139.