Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Λογικης και Επιστημης Υπολογισμων

Πολυπλοκότητα Ισορροπιών σε Βεβαρυμένα Παίγνια

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μουζάκης Νικόλαος

Επιβλέπων: Δημήτριος Φωτάκης
Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, 20/11/2019

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Πολυπλοκότητα Ισορροπιών σε Βεβαρυμένα Παίγνια

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μουζάκης Νικόλαος

Επιβλέπων: Δημήτριος Φωτάκης
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20/11/2019.

.....	.....	.....
Δημήτριος Φωτάκης	Αριστείδης Παγουρτζής	Ευάγγελος Μαρκάκης
Αν. Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ο.Π.Α.

.....
Μουζάκης Νικόλαος
(Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών Ε.Μ.Π.)

Μουζάκης Νικόλαος, 2019

1

Περίληψη

Σε πολλές περιπτώσεις ενδιαφερόμαστε για προβλήματα τα οποία, παρόλο που είναι δύσκολο να λυθούν καθολικά, μπορούν να αντιμετωπιστούν με κάποια ευριστική τοπικής βελτιστοποίησης. Το αρχετυπικό παράδειγμα αυτής της κατηγορίας προβλημάτων είναι το πρόβλημα μέγιστης τομής, όπου βρίσκουμε μια ολοένα μεγαλύτερη τομή γραφήματος αλλάζοντας την πλευρά των κόμβων, μέχρι να μην υπάρχει άλλη δυνατή τοπική βελτίωση μέσω τέτοιων κινήσεων. Έχει δειχθεί ότι αυτό το πρόβλημα είναι πλήρες στην κλάση PLS, που περιέχει όλα τα προβλήματα διακριτής τοπικής βελτιστοποίησης.

Στην παρούσα εργασία εξετάζουμε μια απλούστερη εκδοχή του παραπάνω προβλήματος, εμπνευσμένοι από τις συνδυαστικές ιδιότητες των βεβαρυμένων παίγνιων συμφόρησης. Σε αυτό το πρόβλημα ο γράφος έχει βάρη στους κόμβους αντί για τις ακμές. Χρησιμοποιώντας και τροποποιώντας υπάρχουσες αναγωγές στην κλάση PLS, δείχνουμε ότι το απλούστερο αυτό πρόβλημα μας είναι επίσης πλήρες στην κλάση αυτή. Επιπλέον, εξετάζουμε την πολυπλοκότητα εύρεσης ισορροπιών σε συγκεκριμένες μορφές βεβαρυμένων παίγνιων συμφόρησης. Ειδικότερα, αποδεικνύουμε ότι σε δίκτυα με τοπολογία σε σειρά και παράλληλα η εύρεση αμιγούς ισορροπίας είναι PLS πλήρης. Ακόμα, εφαρμόζοντας την υπολογιστική πολυπλοκότητα του προβλήματος μέγιστης τομής με βεβαρυμένους κόμβους, δείχνουμε ότι σε δίκτυα συμφόρησης με πολλές αρχές και προορισμούς η εύρεση ισορροπίας είναι επίσης PLS πλήρης, ακόμα και όταν οι συναρτήσεις κόστους είναι ταυτοτικές.

Λέξεις-κλειδιά: πολυπλοκότητα αναζήτησης, PLS, μέγιστη τομή, τοπική αναζήτηση, παίγνια συμφόρησης

Abstract

In many settings we are interested in problems that, despite being hard to solve globally, admit some local optimization heuristic that can offer locally optimal solutions. The archetypal example of this class of problems is the MaxCut problem under the flip neighborhood, where we can incrementally improve the value of the cut by flipping nodes. However, it has been shown that calculating the result of this process a priori is at least as hard as solving any other local optimization problem, i.e. it is PLS-hard.

In this thesis, we consider a variation of the above problem, named NodeMaxCut, inspired by the combinatorial structure of weighted congestion games, where the MaxCut graph is instead node-weighted rather than edge weighted. We show that, surpisingly, this problem is still PLS-hard despite its still simple structure, through an involved reduction from the canonical PLS problem CircuitFlip. Our reduction is based on and extends previous work on similar reductions. Secondly, we consider certain forms of weighted congestion games, namely series-parallel weighted congestion games with linear delay functions, as well as weighted multi-commodity congestion games with identity delay functions. The latter is obtained as an application of the PLS hardness of NodeMaxCut. Our work adds a new PLS-complete problem to the literature.

Keywords: search complexity, PLS, local max cut, local search, congestion games

Ευχαριστίες

Καταρχάς, όσον αφορά το περιεχόμενο αυτής της διπλωματικής, ευχαριστώ θερμά όλους τους ανθρώπους των οποίων οι ιδέες και καθοδήγηση συνετέλεσαν στη δημιουργία της. Ειδικότερα, ευχαριστώ τους Στρατή Σκουλάκη, Θανάση Λιανέα και κ.Φωτάκη χάρις στους οποίους πολλά από τα περιεχόμενα αποτελέσματα είναι δυνατά. Χωρίς τη συμβολή και καθοδήγηση τους το ερευνητικό κομμάτι της εργασίας δε θα είχε φτάσει σε αυτό το σημείο.

Σε προσωπικό επίπεδο, ευχαριστώ τους καθηγητές και διδακτορικούς φοιτητές του Εργαστηρίου Λογικής και Επιστήμης Υπολογισμών για τις γνώσεις που προσφέρουν όλα αυτά τα χρόνια, καθώς και την αγάπη για την επιστήμη υπολογιστών που καλλιεργούν. Συγκεκριμένα, ευχαριστώ τους κ. Ζάχο, κ. Φωτάκη και κ. Παγουρτζή. Το μεράκι για την επιστήμη και η βαθιά περιέργεια για τα μυστήρια του υπολογισμού που εμπνέουν όλα τα μέλη του CORELAB θα με συνοδεύουν σε όλα τα επόμενα μου βήματα.

Τέλος, ευχαριστώ τους Βαρδή, Παναγιώτη, Στρατή, Θανάση, Λουκά, Νίκο και Στέλιο για όλη τη συνεργασία, υποστήριξη, συζητήσεις, ιδέες και παρέα στο ΜΟΠ τις δύσκολες στιγμές της έρευνας.

Εκτεταμένη Ελληνική Περίληψη

Σε αυτή την ενότητα παρουσιάζουμε συνοπτικά το περιεχόμενο αυτή της εργασίας στην Ελληνική γλώσσα. Αρχικά, παρουσιάζουμε τις κλάσεις πολυπλοκότητας αναζήτησης με τις οποίες ασχολούμαστε, καθώς και το βασικό πρόβλημα της μέγιστης τομής (MaxCut). Επιπλέον, παρουσιάζουμε το βασικό τεχνικό πρόβλημα αυτής της εργασίας, το πρόβλημα μέγιστης τομής με βεβαρυμένους κόμβους, καθώς και ένα σκιαγράφημα της απόδειξης της απόδειξης πολυπλοκότητας ισορροπιών του. Τέλος, εισάγουμε τον αναγνώστη στα αποτελέσματα εύρεσης αμιγών ισορροπιών σε παίγνια συμφόρησης και δείχνουμε μερικά θεωρήματα για την πολυπλοκότητα αναζήτησης όταν οι παίκτες έχουν βάρη.

Πολυπλοκότητα αναζήτησης

Οι ισορροπίες Nash ενός παιγνίου αποτελούν καταστάσεις όπου κανένας παίκτης δεν έχει κέρδος αν αλλάξει μονομερώς την στρατηγική του. Αποτελούν πολύ φυσιολογικές καταστάσεις που δημιουργούνται σε οποιοδήποτε σύστημα με εγωιστικούς παίκτες και, ως εκ τούτου, αποτελούν αντικείμενο σημαντικής έρευνας για πολλά χρόνια. Το πρόβλημα με πολλές από αυτές τις έννοιες στρατηγικής ισορροπίας είναι ότι είναι γενικά δύσκολο να βρεθούν σε πολυωνυμικό χρόνο, πράγμα που ισχύει ιδιαίτερα για τις αμιγείς ισορροπίες. Παρόλο που σε πολλά παίγνια η ύπαρξη τέτοιων καταστάσεων είναι δεδομένη (μέσω επιχειρημάτων δυναμικού μεταξύ άλλων), η εύρεση τους εξακολουθεί να είναι δύσκολη.

Το παράδοξο προβλημάτων που έχουν πάντα λύση, ενώ ταυτόχρονα η λύση αυτή είναι δύσκολο να βρεθεί έχει οδηγήσει τους ανθρώπους να ορίσουν σχετικές κλάσεις πολυπλοκότητας. Σε αυτές τις περιπτώσεις η πιο διαδεδομένη και ευρέως χρήσιμη κλάση NP δεν μπορεί να βοηθήσει, καθώς η απόφαση για αντικείμενα που μπορεί να μην υπάρχουν και η εύρεση αντικειμένων που πάντα υπάρχουν έχουν σημαντικές διαφορές. Συγκεκριμένα, τα προβλήματα που έχουν πάντα εύκολα ελέγξιμες λύσεις λέγονται ότι ανήκουν στην TFNP, που σημαίνει Total Function Non-deterministic Polynomial. Η κλάση αυτή περιέχει όλα τα δυνατά προβλήματα αναζήτησης. Ωστόσο, κανένα πρόβλημα αυτής της κλάσης δεν μπορεί να είναι πλήρες, καθώς η κλάση είναι σημασιολογικά ορισμένη αντί για συντακτικά. Αυτό σημαίνει ότι και μόνο η απόφαση για το αν ένα πρόβλημα έχει λύση ισοδυναμεί με απόφαση για σημασιολογική ιδιότητα που είναι μη αποφασίσιμο πρόβλημα.

Για αυτό το λόγο, έχουν οριστεί ορισμένες υποκλάσεις της TFNP, οι οποίες επιδέχονται πλήρη προβλήματα. Συγκεκριμένα, με βάση το θεώρημα που εγγυάται την ύπαρξη λύσης σε κάθε περίπτωση, ορίζεται και μια αντίστοιχη υποκλάση της TFNP. Ο ορισμός αυτός είναι πλέον συντακτικός, σύμφωνα με τον οποίο ένα πρόβλημα είναι πλήρες στην αντίστοιχη κλάση αν και μόνο αν είναι τουλάχιστον τόσο δύσκολο όσο η εύρεση της λύσης του αντίστοιχου προβλήματος. Για παράδειγμα, μερικές από τις πιο σημαντικές τέτοιες υποκλάσεις προβλημάτων αναζήτησης είναι οι κλάσεις PPP, PPA, PPAD και PLS, μεταξύ άλλων. Στις επόμενες παραγράφους περιγράφουμε συνοπτικά τις σημαντικές αυτές κλάσεις.

Η κλάση PPP περιέχει τα προβλήματα εντοπισμού λύσης που υπάρχει λόγω επιχειρήματος περιστεροφωλιάς. Δεδομένου ενός κυκλώματος με ίσο αριθμό δυφίων εισόδου και εξόδου, το βασικό πρόβλημα αυτής της κλάσης ζητάει δυο εισόδους που είτε οδηγούν στο ίδιο αποτέλεσμα, είτε μια είσοδο που οδηγεί στο μηδενικό αποτέλεσμα. Η ύπαρξη ενός από τα δυο είδη λύσεων είναι εγγυημένη από την αρχή περιστεροφωλιάς. Προβλήματα σε αυτή τη κλάση αφορούν συχνά τη δυσκολία εύρεσης συγκρούσεων σε συναρτήσεις κατακερματισμού.

Η κλάση PPA περιέχει προβλήματα όπου κατά μια έννοια ζευγαρωμένα μεταξύ τους και αναζητούμε το άλλο άκρο ενός τέτοιου "ζευγαρώματος". Πιο συγκεκριμένα το βασικό πλήρες πρόβλημα αυτής της κλάσης δίνει ένα γράφημα με βαθμούς κορυφών 1 ή 2 και ζητάει, δεδομένης μιας κορυφής βαθμού 1, άλλη μια τέτοια κορυφή. Η μαθηματική αρχή πίσω από αυτή την κλάση είναι το λήμμα της χειραψίας, ύπαρξη περιττής κορυφής σε γράφημα με άρτιες κορυφές σημαίνει ύπαρξη άλλης τέτοιας κορυφής.

Η κλάση PPAD αποτελεί την τομή των δυο παραπάνω κλάσεων και αφορά στην εύρεση της άλλη περιττής κορυφής όταν όμως το γράφημα έχει την παραπάνω ιδιότητα ότι είναι πλέον κατευθυνόμενο. Με αναγωγές μέσω του λήμματος Sperner σε αυτή την κλάση έχουμε την πληρότητα ορισμένων συνεχών θεωρημάτων ύπαρξης σταθερού σημείου.

Η κλάση PLS είναι η πιο μελετημένη από της κλάσεις αναζήτησης, και ιστορικά αυτή που ορίστηκε πρώτη [JPY88]. Περιέχει όλα τα προβλήματα όπου μπορούμε να κάνουμε κάποιου είδους τοπική βελτίωση αλλεπάλληλα, μέχρι που να φτάσουμε σε κάποιο τοπικό ακρότατο που δεν επιδέχεται περαιτέρω βελτιώσεις. Η εύρεση τυχόν τέτοιου τοπικού ελαχίστου είναι το πλήρες πρόβλημα της PLS. Επειδή η κλάση PLS διαδραματίζει πρωταγωνιστικό ρόλο στα θεωρήματα πληρότητας που δείχνουμε παρακάτω, αφιερώνουμε μια ενότητα για να παρουσιάσουμε ορισμούς, θεωρήματα, αναγωγές και προβλήματα σχετικά με την PLS. Για

περισσότερα παραπέμπουμε τον αναγνώστη στις εργασίες των [AAL03],[MAK07].

Πολυωνυμική Τοπική Αναζήτηση

Ο αρχικός ορισμός που δόθηκε για την κλάση αυτή από τους [JPY88] είναι ο παρακάτω.

Ορισμός 1. *Ένα πρόβλημα L στην κλάση PLS αποτελείται από ένα σύνολο στιγμιοτύπων D_L, και από ένα σύνολο λύσεων F_L για κάθε στιγμιότυπο. Σε κάθε λύση s αντιστοιχεί ένα κόστος $c_L(s,x) \in \mathbb{N}$, ένα σύνολο $N(s,x) \subset F_L(x)$ που αποκαλείται η γειτονιά του s καθώς και δυο αλγόριθμοι A_L, C_L. Ο πρώτος επιστρέφει μια τυχαία λύση ενός στιγμιότυπου, ενώ ο δεύτερος επιστρέφει είτε μια λύση $s' \in N(s,x)$ με $c_L(s',x) < c_L(s,x)$ είτε επιστρέφει ότι η λύση είναι τοπικά βέλτιστη*

Με άλλα λόγια, στην PLS ανήκουν όλων των ειδών τα προβλήματα που μπορεί να εφαρμόσει κανείς κάποια αλληλουχία τοπικών βελτιώσεων. Για παράδειγμα, ο γραμμικός προγραμματισμός είναι ένα τέτοιο πρόβλημα, αν σκεφτεί κανείς το simplex pivoting που μας δίνει πάντα μια λίγο καλύτερη λύση. Κατ'εξαίρεση, στο συγκεκριμένο πρόβλημα έχουμε κυρτότητα οπότε η τοπικά βέλτιστη αυτή λύση είναι και καθολικά βέλτιστη. Πέρα από αυτό, προβλήματα συνδυαστικής βελτιστοποίησης που επιδέχονται τοπική αναζήτηση συναντάμε σε πολλούς τομείς.

Το θέμα είναι ότι κάποια από αυτά τα προβλήματα είναι πολύ πιο δύσκολο να λυθούν από άλλα. Για παράδειγμα, το πρόβλημα της τοπικά μεγιστοτικής τομής είναι πολύ πιο δύσκολο από το πρόβλημα του γραμμικού προγραμματισμού, το οποίο επιδέχεται πολυωνυμικά υπολογίσιμες λύσεις. Για αυτό το λόγο, γίνεται χρήση αναγωγών μεταξύ των προβλημάτων PLS ώστε να έχουμε αυστηρές αποδείξεις ότι ένα πρόβλημα τοπικής βελτιστοποίησης είναι πιο δύσκολο από ένα άλλο.

Ορισμός 2. *Ένα πρόβλημα A PLS-ανάγεται σε ένα πρόβλημα B αν υπάρχουν δυο αλγόριθμοι f,g ώστε ο f να στέλνει στιγμιότυπα x του A σε στιγμιότυπα $f(x)$ του B, ο g στέλνει ζεύγη (λύση του $f(x)$,x) σε λύσεις του x, και αν s είναι τοπικό βέλτιστο του B τότε το $g(s,x)$ είναι τοπικό βέλτιστο του x.*

Η σημαντικότερη ιδιότητα αυτών των αναγωγών είναι ότι διατηρούν τα τοπικά ελάχιστα. Δηλαδή αν μετατρέψουμε μέσω μιας τέτοιας αναγωγής ένα πρόβλημα σε ένα άλλο, το οποίο λύσουμε, τότε η λύση που θα πάρουμε για το αρχικό πρόβλημα είναι πάλι τοπικό ελάχιστο. Αυτό κάνει τις αναγωγές αυτές αρκετά πιο περίπλοκες από τις κλασσικές αναγωγές NP.

Ωστόσο, δεν έχουμε ορίσει ακόμα ποια από αυτά τα προβλήματα είναι δύσκολα και ποια όχι. Όπως και στη θεωρία της NP πληρότητας, υπάρχουν πλήρη προβλήματα για αυτή την κλάση στα οποία έχει αποδειχθεί ότι όλα τα προβλήματα της PLS ανάγονται. Συγκεκριμένα, το αρχετυπικό πλήρες πρόβλημα της PLS είναι το πρόβλημα CircuitFlip.

Ορισμός 3. *Στο* CircuitFlip *έχουμε ως στιγμιότυπο ένα κύκλωμα χωρίς feedback το οποίο αποτελείται μόνο από πύλες AND/OR και έχει n εισόδους και m εξόδους. Το κόστος μιας λύσης είναι ο δυαδικός αριθμός που αναπαρίσταται από τα bit της εξόδου. Οι γειτονικές λύσεις μιας λύσης είναι οι δυαδικοί αριθμοί που προκύπτουν με την αλλαγή ακριβώς ενός bit στην είσοδο.*

Μπορεί να αποδειχθεί το παρακάτω θεώρημα για τη δυσκολία αυτού του προβλήματος σε σχέση με όλα τα προβλήματα τοπικής βελτιστοποίησης. [JPY88].

Θεώρημα 1. *Όλα τα προβλήματα στην κλάση PLS ανάγονται στο* CircuitFlip.

Επειδή αυτό το πρόβλημα, παρόλο που είναι πολύ ισχυρό δεδομένης της δυνατότητας του να περικλείει όλα τα προβλήματα της PLS, είναι ιδιαίτερα τεχνητό, κυρίως λόγω του ότι περιέχει κυκλώματα στην περιγραφή του, χρειαζόμαστε κάποιο πιο απλό πρόβλημα για να έχουμε μια γενική θεωρία PLS-πληρότητας. Αυτό επιτυγχάνεται με ένα από τα πιο γνωστά PLS-πλήρη προβλήματα, το πρόβλημα της μέγιστης τομής.

Συγκεκριμένα στο MaxCut κάθε στιγμιότυπο είναι ένα γράφημα κομμένο σε δυο πλευρές (cut). Το κόστος (ή κέρδος) είναι το συνολικό βάρος των ακμών που διασχίζουν την τομής. Γειτονικές λύσεις αποκτούνται όταν αλλάζουμε μια κορυφή από μια πλευρά σε μια άλλη, και τοπικό βέλτιστο έχουμε όταν καμία αλλαγή κορυφής από μόνη της δε μπορεί να αυξήσει το βάρος της τομής.

Παρόλο που το παραπάνω πρόβλημα φαίνεται σχετικά απλό, έχει αποδειχθεί ότι [Sch91] ακόμα και το CircuitFlip ανάγεται σε αυτό. Δηλαδή CircuitFlip $\leq_P$ *LS*MaxCut. Πράγματι, λόγω της απλότητας του αυτό το πρόβλημα χρησιμοποιείται ευρέως για αποδείξεις PLS-πληρότητας, αντί για περίπλοκες αναγωγές στο CircuitFlip.

Ένα από τα προβλήματα που απαντάμε στις επόμενες ενότητες αφορά στην πληρότητα του προβλήματος όταν έχουμε βάρη στους κόμβους και όχι στις ακμές. Όπως το MaxCut είναι πολύ χρήσιμο, ειδικά για αναγωγές σε παίγνια συμφόρησης, το πρόβλημα με βάρη στους κόμβους χρησιμεύει ιδιαίτερα σε PLS-αναγωγές όπου οι παίκτες έχουν βάρη.

Μέγιστη τομή με βεβαρυμένους κόμβους

Σε αυτή την ενότητα παρουσιάζουμε μια εποπτεία της απόδειξης της PLS-πληρότητας του προαναφερθέντος προβλήματος. Ειδικότερα, τονίζουμε ότι λόγω της βαρύτητας του δε μπορεί να αναχθεί το MaxCut άμεσα σε αυτό, πράγμα που δείχνει ότι το πρόβλημα είναι αρκετά πιο θεμελιώδες από το MaxCut. Στα παρακάτω θα αναφερόμαστε στο πρόβλημα ως NodeMaxCut.

Η καρδιά των τεχνικών εργαλείων που απαιτούνται για την απόδειξη αυτή βρίσκονται στις συσκευές που παρουσιάστηκαν αρχικά στο [Sch91] και επαναχρησιμοποιήθηκαν αργότερα για παρόμοιες αναγωγές [GS10],[ET11]. Οι συσκευές αυτές αποτελούν υπογραφήματα με την ιδιότητα να προσομοιάζουν στη συμπεριφορά πυλών NOR. Ειδικότερα, κάθε τέτοιο γράφημα έχει κάποιους εξωτερικούς κόμβους, που αποκαλούμε ακροδέκτες. Όταν αυτοί έχουν κάποιες συγκεκριμένες τιμές τότε η συσκευή συμπεριφέρεται με δυο τρόπους: Στη μια περίπτωση οι κόμβοι εισόδου δεν έχουν πλέον καμία επίδραση από τους εσωτερικούς κόμβους, ενώ στη δεύτερη περίπτωση οι κόμβοι εξόδου έχουν μεγάλο κέρδος αν πάρουν την τιμή που αντιστοιχεί στο NOR των εισόδων. Έχουν δηλαδή μια λειτουργία "εισόδου" και μια λειτουργία "υπολογισμού", το οποίο ελέγχεται από κάποιους κόμβους ελέγχου.

Η βασική δυσκολία στην αναγωγή είναι να κάνουμε αυτές τις συσκευές να λειτουργήσουν σε στιγμιότυπα που έχουμε βάρη μόνο σε κόμβους. Συγκεκριμένα, δημιουργούμε μερικές κατασκευές σε γραφήματα βεβαρυμένων κόμβων που έχουν τις ιδιότητες που θέλουμε και μας επιτρέπουν να ελέγξουμε τους κόμβους ελέγχου, καθώς και να επιτύχουμε σύγκριση των εξόδων των κυκλωμάτων. Στις επόμενες παραγράφους επιχειρούμε να περιγράψουμε τη λειτουργία αυτών των κατασκευών.

Η πρώτη κατασκευή αφορά των έλεγχο των ακροδεκτών που ελέγχουν τη λειτουργία της συσκευής NOR. Εδώ η δυσκολία είναι ότι οι ακροδέκτες έχουν πολύ μεγάλο βάρους κόμβου σε σχέση με το βάρος της εξόδου, που θα πρέπει να τους ελέγξει. Ωστόσο, αυτοί οι ακροδέκτες συνορεύουν με κόμβους πολύ μικρού βάρους, που σημαίνει ότι μπορούμε να τους ελέγξουμε με οσοδήποτε μικρού βάρους κόμβους, αρκεί να μπορέσουμε να κάνουμε αυτούς τους μικρούς κόμβους να πάνε στη σωστή θέση κατ' αρχάς. Για να το επιτύχουμε αυτό παρατηρούμε ότι η αναλλοίωτη που μειώνεται κατά μήκος μιας αλυσίδας αιτιότητας κόμβων, δηλαδή όταν λέμε ότι κάποιος κόμβος θα έχει αναγκαστικά κάποια τιμή εξαιτίας ενός άλλου κόμβου κλπ, είναι το γινόμενο βάρος κόμβου επί το bias που έχει ο κόμβος. Με άλλα λόγια ένας μικρός κόμβος με μικρό bias μπορεί να επηρεάσει έναν μεγάλο κόμβο με πολύ μικρό bias. Εφαρμόζοντας αυτό το τέχνασμα κατορθώνουμε να προσομοιώνουμε κατευθυνόμενες ακμές που έχουν αυθαίρετο βάρος. Φυσικά, αυτό δεν αρκεί γιατί αλλιώς θα μπορούσαμε να έχουμε απευθείας αναγωγή από το MaxCut.

Το δεύτερο σημαντικό κομμάτι της αναγωγής που λείπει για την κατανόηση της είναι το ότι για να επιτύχουμε τόσο σύγκριση όσο και έλεγχο σφαλμάτων, όπως γίνεται στην αναγωγή του MaxCut [Sch91], πρέπει να χρησιμοποιήσουμε μόνο έναν κόμβο σύγκρισης. Επιπλέον, αυτός ο κόμβος πρέπει να ενωθεί με όχι μόνο τον τελευταίο ακροδέκτη ελέγχου, αλλά με όλους. Ο λόγος είναι ότι από τη μια αν ενωνόταν μόνο με τον τελευταίο τότε το bias σε περίπτωση λάθους θα ήταν παρόμοιο με το βάρους του ελαχίστου bit που δε θα αρκούσε για να αλλάξει την κατάσταση του κόμβου σύγκρισης παρά το λάθος, ενώ από την άλλη αν το ενώσουμε με όλους τότε μπορεί το λάθος να έχει επίδραση στον κόμβο σύγκρισης ίση με τη επίδραση του λάθος. Όταν είναι ίσα τότε μπορεί το λάθος να μην υπερνικά την λανθασμένη σύγκριση αλλά, παίρνοντας περιπτώσεις, δείχνουμε ότι τότε το κύκλωμα που χάνει τη σύγκριση είναι ούτως ή άλλως μικρότερο (αφού βγάζει αναγκαστικά 0 εκεί που το άλλο βγάζει 1 για να υπερνικά το λάθος).

Με αυτές τις δυο παραπάνω τεχνικές διαισθήσεις μπορούμε να κάνουμε τις NOR συσκευές να λειτουργήσουν σε ένα περιβάλλον με βεβαρυμένους κόμβους. Η απόδειξη είναι αρκετά τεχνική και παρουσιάζεται στο αγγλικό κείμενο παρακάτω. Τονίζουμε ότι η εκτενής παρουσίαση της αναγωγής στο αγγλικό κείμενο αποτελεί πρακτικά προσεκτική εφαρμογή των παραπάνω δυο ιδεών σε ορισμένα σημεία. Εν τέλει καταλήγουμε στο παρακάτω θεώρημα:

Θεώρημα 2. *Το* NodeMaxCut *είναι PLS-πλήρες.*

Βεβαρυμένα Παίγνια Συμφόρησης

Η σημαντικότερη εφαρμογή της πληρότητας αυτού του προβλήματος είναι στο να αποδεικνύουμε αποτελέσματα σχετικά με τη δυσκολία ισορροπιών σε παίγνια συμφόρησης. Σε προβλήματα συμφόρησης έχουμε διαφορετικούς παίκτες οι οποίοι επιλέγουν πόρους και δημιουργούν συμφόρηση σε αυτούς. Επειδή οι παίκτες συμπεριφέρονται εγωιστικά η διαδικασία μοιάζει με τοπική βελτιστοποίηση όταν υπάρχει γενική ποσότητα που ελαττώνεται σε κάθε κίνηση παίκτη. Για αυτό τον λόγο η θεωρία της PLS πληρότητας χρησιμοποιείται συχνά για να δείξει δυσκολία εύρεσης ισορροπιών Nash σε τέτοια παίγνια.

Ορισμός 4. *Ένα παίγνιο συμφόρησης* $G = \{N, F, (A_i)_{i\in[N]}, (d_f), f \in F\}$ *όπου [N] αναπαριστά το σύνολο των παικτών, F αναπαριστά το σύνολο των πόρων,* $A_i \subset 2^F$ *αναπαριστά τις στρατηγικές του παίκτη i και* $d_f : N \to Z$ *τη συνάρτηση κόστους που σχετίζεται με τον πόρο f.* $a = (a_1, ..., a_N)$ *είναι το προφίλ στρατηγικών όπου ο παίκτης i επιλέγει την στρατηγική* $a_i \in A_i$*. Για ένα προφίλ a η συμφόρηση* $n_{f(a)}$ *ενός πόρου f ορίζεται ως* $f(a) = |\{i \in [N] | f \in a_i\}|$*. Το κόστος κάθε παίκτη είναι το σύνολο των κοστών των πόρων στην επιλεγμένη στρατηγική του* $c_i(a) = \sum_{(f\in a_i)} d_f(n_f(a))$.

Στα γενικά παίγνια συμφόρησης έχει δειχθεί από τον Rosenthal ότι υπάρχει ένα δυναμικό που συσσωρεύει τις βελτιώσεις όλων των παικτών σε κάθε κίνηση [Ros73].

Θεώρημα 3. $F(S) = \sum_{f\in F} \sum_{k=1}^{N_f} d_f(k)$ *αποτελεί ακριβές δυναμικό για τα παίγνια συμφόρησης.*

Το παραπάνω σημαίνει ότι τα παίγνια συμφόρησης μπορούν να αναλυθούν ως προβλήματα τοπικής βελτιστοποίησης, καθώς εγωιστική κίνηση παίκτη ισοδυναμεί με τοπική βελτίωση. Φυσικά, σε περιπτώσεις που υπάρχουν ισορροπίες Nash αλλά δεν υπάρχει δυναμικό και η ύπαρξη τους βεβαιώνεται με κάποιο άλλο επιχείρημα, πχ λεξικογραφικό, η θεωρία της PLS δεν μπορεί να βοηθήσει.

Στη γενικότερη περίπτωση, η εύρεση ισορροπιών έχει αποδειχθεί ότι είναι PLS-δύσκολη από τους [FPT04]. Εκεί αποδεικνύεται ότι τα παίγνια αυτά στην πιο αφηρημένη τους μορφή είναι ακόμα πιο δύσκολα από την εύρεση ισορροπιών στη μέγιστη τομή, μέσω μιας αναγωγής από το MaxCut σε αυτά. Επιπλέον, δείχνουν ότι όταν το παίγνιο συμφόρησης έχει τη μορφή δικτύου, τότε είναι εύκολο να βρούμε αμιγείς ισορροπίες όταν όλοι οι παίκτες έχουν τον ίδιο προορισμό και αρχή, ενώ είναι PLS-δύσκολο όταν έχουμε πολλούς προορισμούς. Η δεύτερη απόδειξη είναι αρκετά περίπλοκη, ανάγοντας από το CircuitFlip.

Αργότερα, οι Vocking et al έδειξαν ότι τα παίγνια σε δίκτυα είναι PLS-δύσκολα ακόμα και όταν οι συναρτήσεις κόστους είναι γραμμικές. [ARV08]. Με το παραπάνω αποτέλεσμα, η πολυπλοκότητα των αμιγών ισορροπιών σε παίγνια συμφόρησης είναι πρακτικά κλειστό πρόβλημα.

Ωστόσο, μια ενδιαφέρουσα παραλλαγή των κλασσικών παίγνιων συμφόρησης είναι όταν έχουμε διαφορετικά βάρη στους παίκτες. Το μοντέλο αυτό, αν και μικρού ενδιαφέροντος από μόνο του, απέκτησε μεγάλη σημασία όταν δείχτηκε από τους [FKS05a] ότι για γραμμικές συναρτήσεις κόστους υπάρχει βεβαρυμένο δυναμικό. Συγκεκριμένα, έδειξαν το παρακάτω θεώρημα.

Θεώρημα 4. $\sum_{i\in N} w_i \sum_{e\in s_i} (a_e * w_i + b_e) + \sum_{e\in E} s_e * (a_e * s_e + b_e)$ *είναι δυναμικό για βεβαρυμένα παίγνια συμφόρησης με γραμμικά κόστη.*

Επομένως, άμεσα έχουμε ότι η κατηγορία των παίγνιων με βεβαρυμένους παίκτες είναι μια κατηγορία παιγνίων που πρέπει να αναλυθεί όπως και τα κλασσικά παίγνια συμφόρησης. Το πρόβλημα, ως τώρα, είναι ότι τα μόνα αποτελέσματα γνωστά για PLS-πληρότητα βεβαρυμένων παίγνιων πηγάζουν από απλές γενικεύσεις των παραπάνω αποτελεσμάτων με τον προφανή τρόπο, δηλαδή θέτοντας όλους τους παίκτες με το ίδιο βάρος. Φυσικά, αυτό δεν αποκαλύπτει την ακριβή επίδραση που έχει η ύπαρξη βαρών στους παίκτες, η οποία όπως δείχνουμε σε αυτή την εργασία, είναι μεγάλη.

Συγκεκριμένα, αποκαλύπτουμε ότι όταν έχουμε βεβαρυμένους παίκτες τότε ο υπολογισμός ισορροπίας είναι PLS-δύσκολος, ακόμα και στις περιπτώσεις που το δίκτυο έχει τη μορφή σύνδεσης σε σειρά και παράλληλα. Σε αυτές τις τοπολογίες έχει δειχτεί ότι η ισορροπία είναι εύκολο να βρεθεί με μια απλή άπληστη στρατηγική [FKS05b]. Η δικιά μας αναγωγή, που αποδεικνύει το παρακάτω θεώρημα, χρησιμοποιεί με καίριο τρόπο τα βάρη των παικτών έτσι ώστε να προσομοιώσει ένα στιγμιότυπο του MaxCut.

Θεώρημα 5. *Τα βεβαρυμένα παίγνια συμφόρησης σε συνδεσμολογία σειρά και παράλληλα έχουν PLS-δύσκολες αμιγείς ισορροπίες.*

Ένα άλλο ερώτημα που εγείρεται σχετικά με την επίδραση των βαρών στη δυσκολία των ισορροπιών αφορά στο τι γίνεται αν βγάλουμε όλους τους συντελεστές από τις γραμμικές συναρτήσεις κόστους και αφήσουμε βάρη μόνο στους παίκτες. Στην εργασία αυτή δείχνουμε ότι, ακόμα και τότε, τα βάρη των παικτών αρκούν για να κάνουν τις ισορροπίες PLS-δύσκολες.

Θεώρημα 6. *Τα βεβαρυμένα παίγνια συμφόρησης με ταυτοτικά κόστη έχουν PLS-δύσκολες αμιγείς ισορροπίες.*

Το παραπάνω θεώρημα προκύπτει μέσω μιας αναγωγής από το πρόβλημα που ορίσαμε και αποδείξαμε PLS-δύσκολο, το NodeMaxCut. Αυτό είναι ένα από τα προτερήματα του καθώς το απλό MaxCut δε θα μπορούσε να απεικονίσει τη δυσκολία αυτού του προβλήματος.

Contents

Chapter 1

Introduction

Local search is ubiquitous in nature. No agent has omniscient access to all information of its environment, which leads it to make short-term, myopic decisions based on its locally available information. A prime example of this is evolution ([Kaz19]), considering how survival of the fittest causes the adaptivity of a species to be optimized. Since there is no central authority instructing the system on which characteristics are ideal, all improvements in biological organisms happen through an iterative locally improving process, i.e. the best immediate characteristic is picked each time.

The deep power of local search has inspired humans to apply it on solving important optimization problems. We often find ourselves in settings where obtaining all possible information and making a globally optimal decision is infeasible, such as picking what to wear, in which case we might iterate on an existing wardrobe, until we settle on an attire where no small improvements can be made. More importantly, however, local search heuristics have provided significant help in facing fundamental computational problems where global optimization is provably intractable. One of the most celebrated such success stories was the travelling salesman problem where an efficient tactic for producing "good enough" hamiltonian circuits comes from the local search paradigm. Specifically, one of the historically first such local improvement heuristics, proposed by Croes [Cro58], was simply to start from an arbitrary hamiltonian cycle, and then examine whether swapping any pair of edges would offer a better cycle. In practice this algorithm produces solutions of very high quality as well as terminating fast in most cases. Indeed, one can show that the smoothed complexity of this process is in fact polynomial. Other heuristics for this problem have also been proposed, such as the Kernighan-Lin neighborhood, where the neighborhood has a depth first structure, instead of breadth first. Another domain of application of local search is the celebrated Simplex algorithm. Overall, local search has met huge success in the field of combinatorial optimization and is a powerful technique one may apply in various settings.

However, local search does not show up only in human-devised algorithms. While in an algorithm there is a central authority, a program, that guides the optimization process, seeking to improve some overall cost function, in nature there are multiple agents that only seek to improve their own lot. The only guarantee we have regarding the rationale behind the decisions of these agents is that they will act selfishly and take actions for solely personal gain. There are cases, however, where the actions of these agents happen to contribute to a global quantity. To elaborate, in these multiagent systems there exists a combinatorial invariant that gets increased by an action if and only if that action improves the lot of the actor. Specifically, when we consider systems with this property as games with multiple players, we call them "potential games", since akin to a potential in physics, that global quantity implicitly tracks the progress of the players towards a selfish local optimum. Viewing through this lens, one can see why the natural process of players playing a potential game is computationally equivalent to local search with arbitrary pivot rules.

Central to the study of potential games are games of a special form: i.e. congestion games. Congestion games consist of a set of players, as well as a set of resources that players utilize and induce delay on, according to some cost function. They not only possess a potential function [Ros73] but they in fact capture the entire class of potential games. This is because for every potential game there is a congestion game with the same potential function [MS96], which renders congestion games not only a useful model for examining real-life congestion scenarios, like traffic or job scheduling, but also for obtaining insight on computational aspects of potential local minima.

However, calculating any local minimum of the potential function in congestion games, under the neighborhood where players make selfish moves, has been shown to be PLS-hard [FPT04]. Polynomial Local Search (PLS) is a complexity class [JPY88] meant to capture the hardness of computing the endpoint of a local search iterative procedure. In other words, a PLS-hard problem is at least as hard as any local search problem one can imagine. Hence, calculating such a local optimum, or in game theory terms, a pure nash equilibrium, is unlikely to be possible in polynomial time, unless several hard optimization problems can be efficiently solved. This raises the following interesting question, in the words of Kamal Jain: "If your laptop can't find it, neither can the market". Responses to this divergence between theory and reality (after all supply and demand quickly reaches stability in real world markets!), include the rejection of pure nash equilibria as meaningful notions and replacing them with the much more tractable correlated equilibria, as well as going beyond worst case analysis and considering the smoothed complexity of iterative algorithms. In fact, the latter approach has had certain success stories including the smoothed analysis of max-cut [ABPW17], k-means [AMR11] and the simplex method for LP [ST04].

Of particular interest to us are congestion games with a slight modification. Namely, the players have non-uniform impacts on the cost functions of the resource delays, i.e. the players are weighted. These games are predictably called weighted congestion games. Unlike the unweighted case, however, these games do not admit a potential function (and hence no guarantee of existence of PNE) in the general case [FKS05a]. For the existence of such a potential function to be guaranteed we need further stipulations on the actual cost functions of the resources. In particular, it has been shown that a necessary and sufficient condition for a potential function to exist is that the delays must either be a linear or exponential function of the sum of the weights of the players using them [FKS05a],[PS07]. In either of these cases a potential function can be found.

Similar to unweighted congestion games, pure nash equilibria (equivalently local minima of the potential function) are PLS-hard to find. [ARV08]. However, in all such reductions the players' weights themselves don't play a meaningful role, and in fact these reductions can work even for unweighted players. This leaves open the question of whether placing weights on players does in fact make the problem of finding equilibria fundamentally more difficult from a computational perspective, or the difficulty is in fact implicit into the particular delay functions chosen.

1.1 Technical Contribution

Our main motivation for this work is examining the PLS complexity of pure nash equilibria in player-weighted congestion games. However, unlike previous work on the PNE complexity of congestion games ([FPT04],[ARV08]), we wish to focus on the computational impact of the player weights themselves, which means we concentrate on weighted games where the unweighted version has polynomially computable equilibria.

MaxCut is a prominent PLS-complete problem, based on which a large number of PLS-hardness results can be proven, including several concerning congestion game equilibria. For our purposes in this thesis, MaxCut is insufficient, because all edges have different weights and thus we cant use it in our reductions. For this reason, we define a natural simplification of MaxCut, called NodeMaxCut, where the graph is now node-weighted instead of edge-weighted. In this new problem, players are identified with nodes, each seeking to choose the side of the cut with the least total weight of neighbors present in the same side. This problem does possess a pure nash equilibrium, once one notices that it has a weighted potential, i.e. the weight of the node multiplied by the local improvement. The problem itself is interesting due to its apparent simplicity and succinctness, leading one to think that a polynomial time algorithm should exist for computing a PNE of a NodeMaxCut instance.

However, our main technical contribution of this thesis is to show that this problem is, in fact, PLS-complete. The proof of this result is rather involved using the general framework proposed in [Sch91], but with several new ideas that shed light on how weights alone can provide the necessary complexity through the weighted potential function. Overall, our reduction consists of a complex NodeMaxCut instance graph, whose equilibria would reveal an equilibrium value of any CircuitFlip instance. More specifically, our reduction uses the gadgets proposed by Schaffer and Yannakakis to emulate NOR logic gates in order to show the PLS-completeness of MaxCut. These NOR gadgets have several useful properties that are necessary for constructing and operating an arbitrary boolean circuit in a MaxCut

(or NodeMaxCut in this case) instance. They have been constructed in such a way that a control node, having minimal weight itself, allows the inputs to either change freely or "lock" and calculate the NOR value.

Our reduction needs certain additional constructions in order to make these NOR gadgets work in a NodeMaxCut setting. The first, and most important, of these additions is a "leveraging" gadget that permits the lower weighted nodes of the output of this circuit to influence the heavier control nodes of the NOR gadgets. This is achieved by alternating increasingly heavy and light nodes, so as to augment the ability of the gadget to move a heavy node, without violating the fact that the potential must decrease along this process. The second addition is more complicated and concerns the overall architecture of our construction. It relates to the fact that one cannot properly achieve comparison between the outputs of the Boolean circuit, as well as make sure that only one circuit is active at the time, using weights only on the nodes. For this reason, we have a single *flag* node, unlike previous reductions, e.g. [GS10],[ET11], which have two. Moreover, in our reduction we have to connect all the control variables of the NOR gadgets to the $flag$ node unlike previous reductions. This is because of the restrictive nature of not being able to have weights on each individual edge, unlike existing approaches, which can use any weight on each edge. In particular, connections between the *flag* node and the control variables are not made only once to ensure correctness of the whole circuit, but are instead made for each control node corresponding to each bit of the output. This way the incorrectness of a bit will cause $flag$, though the control node, to ignore its value.

While the PLS-completeness of NodeMaxCut might be of independent interest to readers, our main use for it is to show certain PLS-hardness results, which, without NodeMaxCut, would be difficult to prove. Specifically, we show that multi-commodity weighted congestion games with identity delay functions are PLS-hard. This is an interesting development, since it implies that any meaningful extension of the celebrated restricted links algorithm of [GLMM10] on any larger network would have to solve a PLS-hard problem. To show our result, we use the fact that in the case of large amounts of constant delay (which can be modeled with a very heavy player) players will choose to take the shortest path regardless of other players' actions. By exploiting this, we force the players to take specific paths in a grid-like network, inspired by the construction of [ARV08], and hence emulating a NodeMaxCut instance.

In addition to the above, we also present a novel proof of the PLS-completeness of weighted congestion games played on series-parallel network topologies. This shows that the results of [FKS05b] for series parallel networks cannot be extended to the unrestricted case with arbitrary weights on players. To prove this, unlike the previous cases, we reduce directly from MaxCut. We use the presence of weights on the players as separators to force them to interact in the specific ways they would if they were nodes in a MaxCut instance.

1.2 Organization of this work

In the first chapter we survey progress that has been made so far in the direction of tractability of total search problems. We present general results from the literature from reaserch on inefficient proofs of existence along the last decades.

In the second chapter we focus on the PLS complexity class, which is directly connected to our work. Here, we rigorously define and explain the toolbox we are going to use in further chapter to prove PLS-harndess for our problems.

In the third chapter we fully present the PLS-completeness proof of node weighted max cut, which is the main point of interest of this thesis. The proof is long and convoluted, since it reduces from a Circuit problem, but it possesses certain interesting ideas used for utilizing the presence of a weighted potential instead of an exact one.

In the fourth chapter we introduce the reader to the theory of congestion games and their local search dynamics. We survey the bibliography, concentrating on existing results on the complexity of Pure Nash Equilibria.

In the last chapter, we present our PLS-hardness results for specific classes of weighted congestion games. In particular, we use a simple reduction to induce PLS-hardness through player weights for series-parallel networks. Also, we apply our NodeMaxCut problem to show PLS-hardness for congestion games where the delays are identity functions.

Chapter 2

Complexity of Searching

Suppose you have a very difficult algorithmic problem to solve, which requires you to find some solution. Despite your best efforts that solution appears to be impossible to calculate and no algorithm you try seems to work. For this reason, you turn to the rich field of complexity theory, with the purpose of reducing some known hard problem to your own, thereby proving that's its futile to insist in a solution to your problem. This is because such a solution would imply so much more than simply resolving your problem. It would, in fact, permit the resolution of that much harder problem you reduced from.

This approach of reducing from known hard problems to your own, has in fact found groundbreaking success in the last decades with the theory of NP-completeness. In particular, NP-complete problems are decision problems where even being able to answer whether a solution exists is difficult. An example is the classic SAT problem where given a formula we are required to answer if its satisfiable. It turns out that there a multitude of problems that exhibit this structure of a solution being hard to find, but easy to verify, assuming it even exists.

What happens, however, when the problem you are considering is in fact strictly easier than any NP problem? Specifically, what if, for some reason, you are guaranteed that the solution you are looking for does exist, but you have no clue how to find it. In reality, such problems crop up all across theoretical computer science, and beyond. From the computation of Nash Equilibria to the calculation of fixed points, problems whose solutions we know for certain to exist, and which we could readily identify if presented with, crop up in a variety of places. Because we are guaranteed the existence of at least one solution, the problems of the class NP are not adequate for reducing, leaving it uncertain how we might obtain complexity lower bounds for our problems.

To alleviate this, a subclass of FNP has been identified, namely TFNP, with the purpose of capturing exactly these problems where we are searching for something that we know exists.

In this chapter we will define the basic complexity framework that allows us to capture the difficulty of search problems. Unlike decision problems, where a simple yes/no answer will suffice in search problems we also ask for the certificate that renders an instance of a problem a yes-instance. This further demand allows us to pose interesting questions on problems that might in fact be easier or harder than mere decision.

2.1 From decision to search

The class NP is the class of all decision problems that admit a succinct certificate. We can define a direct function analogue FNP (function nondeterministic polynomial) which consists of the problems or, equivalently, relations where either a short solution can be reported or a negative answer if no such solution exists.

Definition 2.1.1. FNP

A problem in FNP consists of: (a) A set D_L of polynomially recognizable strings (the instances). (b) For each instance, a set of solutions $F_L(x), x \in D_L$ which are polynomially bounded in length with respect to the length $|x|$ of the instance. (c) A binary relation $R(x, y)$ such that $R(x, y) \iff x \in D_L \wedge y \in F_L(x)$

Every problem in NP can be easily transformed into its standard function counterpart, where we demand the certificate as the solution. An immediate question that arises is how much harder is actually

returning the certificate compared to simply deciding about its existence. Maybe there exist problems where locating the certificate would be intractable, even if we had an oracle that decides its existence?

Perhaps unsurprisingly, the answer turns out to be negative. The search versions all NP problems can be reduced to their decision analogues. For instance, FSAT, where we are asked to find a satisfying assignment to a given boolean formula can be solved easily with access to an oracle that tells us whether a formula can be satisfied or not. In particular, assuming that a variable of a formula takes a certain truth value, we can ask the oracle if the rest of the formula can be satisfied and, if not, we can assume the opposite value. By repeating this, we can find the actual satisfying configuration. Moreover, because any search problem can be reduced to FSAT (by modifying the classic NP reductions) and SAT can be reduced to any NP-complete problem, we have that all search versions of NP-complete problems are no harder than their decision version.

Theorem 2.1.1. $FNP = FP \iff NP = P$

We note that the above self-reducibility property only applies to NP-complete problems (since SAT only reduces to those) and not simply NP problems. For instance FACTORING, the problem of determining a divisor and its corresponding search version PRIME, cannot be reduced to each other. In fact, it is known through a breakthrough result of Agrawal et al [AKS04] that determining the primality of a number can be done in polynomial time, but a polynomial algorithm for factoring does not currently exist. If it did, the consequences for modern cryptography would be catastrophic. The actual hardness of FACTORING is, presently, an important open problem.

Furthermore, it was shown by Bellare et al [BG94] that, similar to Ladner's theorem and the existence of NP-intermediate problems, under certain assumptions, then there are NP problems that are easier to decide than provide concrete solutions to.

Theorem 2.1 renders any inquiry into the hardness of searching for the certificates of NP-hard problems meaningless, since we may as well reformulate these questions with respect to NP, which is a well explored class.

The deeper reason, however, that such search problems are so difficult (i.e. as hard as NP) is that the problems we investigate may as well not have any solution forcing us, indirectly, to also answer the question of its existence at the same time when searching. The most interesting search problems are those where, somehow, we are guaranteed that what we are searching for exists. These problems obviously have no meaningful decision analogues and yet capture the true essence of the difficulty of the computation of searching. This class, introduced by Papadimitriou et al [MP91] is named TFNP.

2.2 TFNP

TFNP stands for Total Function Nondetermistic Polynomial. As the name implies, it is the class that encapsulates the difficulty of problems where one is asked to search for something whose existence is guaranteed. In other words, the relations $R(x, y)$ that comprise this class are such that for every instance x there is always a solution y that satisfies.

Definition 2.2.1. [MP91]
TFNP is the class FNP constricted to total relations $R(x, y)$.

Naturally, problems in TFNP should be easier than problems in FNP, since they are only a subset of the latter. Indeed, the fact that we are guaranteed that we can find a solution can only make things easier. However, this totality gives an even stronger indication of the fact that TFNP lies strictly below FNP, with respect to difficulty. Specifically, imagine that there exists a problem in TFNP so fiendishly difficult that, given an oracle that solves it, we can use that oracle to solve NP-hard problems. This would mean that if the instance of any NP-problem is a yes-instance then a call to our TFNP oracle would produce some succinct solution which we would then manipulate to obtain our yes-certificate. Conversely, if the NP-hard problem was a no-instance our oracle would again return a solution (it always returns a solution since its total!) which we would manipulate to obtain our no-answer. But then, one may ask, what if we took that solution that the oracle returned, and used it as a succinct no-certificate? If that were true, the we would be able to construct a way such that any NP-problem can not only admit yes-certificates, but also succinct no-certificates. This would imply that NP=co-NP (and also that the

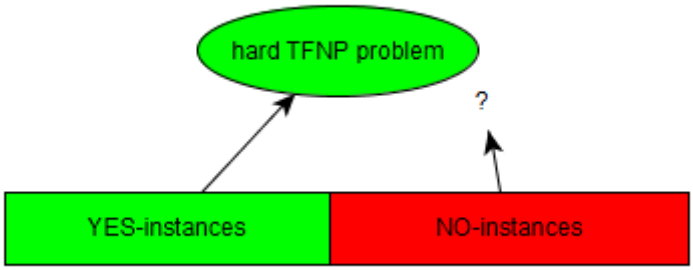

Figure 2.1: Schematic showing NO-instances cannot be matched to any call of the TFNP oracle unless NP=co-NP

polynomial hierarchy collapses to the first level). Hence, unless we concede that all NP problems can also admit no-certificates, we have serious reason to believe that the totality of TFNP problems places them strictly below FNP in difficulty. That is not to say that TFNP problems are easy. There has been a multitude of lower bounds for total search problems showing they are hard (even for quantum computers [Aar06].

On the other hand, we have no serious indication that TFNP is an easy class, i.e. lies within the realm of FP. The problems that are contained in TFNP are, despite the guarantees of the existence of the solution, still hard on their own right. Such problems include finding nash equilibria, calculating stable points of functions or even the notorious problem of factoring, all of which are considered intractable for the tools humanity currently possesses. Then again, there are problems in TFNP that are trivially easy to solve, such as simple algebraic equations. We therefore need to define the complete problems of this class, in order to be able to classify problems as being intrinsically intractable or not. The major difficulty into which one runs while attempting to define the archetypal TFNP problem is that even deciding whether a relation $R(x, y)$ is total is NP-hard. This makes it unlikely for an extremely generic TFNP problem to exist, capable of capturing all total search problems with no additional guarantees on the nature of the relation $R(x, y)$ itself. In other words, TFNP is a semantic class. To that end,i.e. to obtain syntactic definitions, several subclasses of relations were defined by Papadimitriou et al, such that each guarantees the totality of the relation through a simple combinatorial argument. These subclasses are:

- PLS: existence of a sink on a directed graph
- PPA: handshaking lemma
- PPP: pigeonhole principle
- PPAD: directed handshake
- CLS: continuous fixed point

These classes are obtained as syntactic formulations of their corresponding existence theorem. We mention here that in Papadimitriou's original paper [Pap94] a class for the probabilistic existence argument was proposed, and especially for the Lovasz Local Lemma, namely the class PLL. In the meantime, however, it was shown by Moser and Tardos [MT10] that LLL objects can in fact be efficiently constructed which meant this class was in P and hence not interesting. We also note that while the three principal classes (PPA,PPP,PLS) might seem disjoint with no common combinatorial structure, recent work [GP17] has revealed the existence of a unified syntactic class that can in fact capture all of the aforementioned classes, namely PTFNP (Provable TFNP), that possesses a simple complete problem. That problem is called Wrong Proof and, given a circuit that produces the logical steps of a (exponentially sized) proof , i.e. use of axioms or inference rules, and which also produces a false conclusion as its last step, we are asked to find the mistake in the proof. This is nontrivial and in fact it turns out that solving this problem is at least as hard as solving any problem of the known TFNP classes.

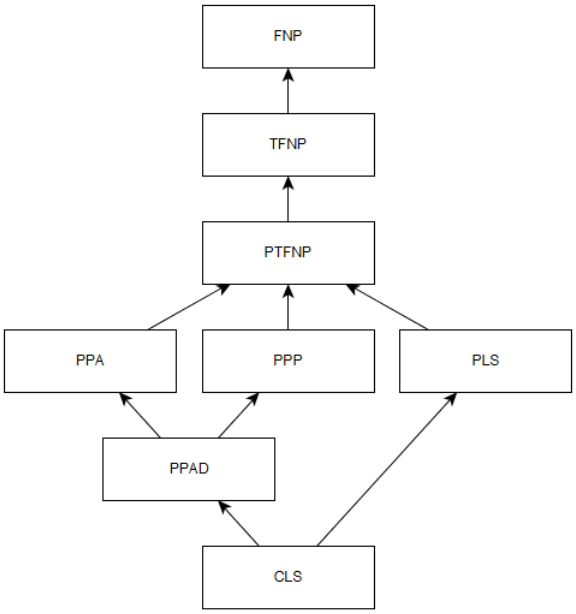

Figure 2.2: Hierarchy of some of the currently known search classes

2.2.1 PPP

PPP (short for Polynomial Pigeonhole Principle) is the total search complexity class meant to capture problems related to the inefficient proof of the pigeonhole principle. In particular, problems in this class require some sort of collision to be found. Naturally, this class is useful for modelling cryptography and hashing related problems.

Its archetypal complete problem is the following problem, called PIGEON. [Pap94]

Definition 2.2.2. Given a circuit C with n input and n outputs, find either an input u such that $C(u) = 0$, or find two inputs u, v such that $C(u) = C(v)$.

Moreover, another subclass closely related to PPP but contained within it is PWPP (polynomial weak pigeon principle) which is the set of all problems polynomially reducible to WEAKPIGEON

Definition 2.2.3. Given a circuit C with n input and less than n outputs, find two inputs u, v such that $C(u) = C(v)$.

In other words, PWPP is the PPP class only constrained to a non-injective setting, forcing a collision to exist.

Despite its very simple description and ease of understanding, the PPP class remained without a natural complete problem not making explicit references to circuits for several decades until the work of [SZZ18] who showed the existence of a natural PPP-complete problem related to lattice cryptography.

A very interesting problem whose completeness status is not yet know is FACTORING. It was shown by [Jeř16] that factoring is both in PPA and (through a randomized reduction) in PPP.

2.2.2 PLS

One of the other major classes characterized by a nonconstructive proof of existence is the PLS (Polynomial Local Search) class. In this case, the profound combinatorial argument used to show the existence of the solution is the presence of a sink in a directed acyclic network. Considering how the rest of this work is deeply intertwined with the PLS class we are going to give only cursory definitions here and defer the detailed description of this class to the next chapter.

The canonical PLS-complete problem is CIRCUITFLIP.

Definition 2.2.4 ([JPY88]). An instance of CircuitFlip can be described as a feedback-free Boolean Circuit made up of AND, OR and NOT gates with n input bits $(x_0, x_1, x_2...x_n)$and m output bits $(y_0, y_1, y_2...y_m)$. The cost $c_L(x)$ is the binary value associated with its output, i.e. $\sum 2^i * y_i$. The neighborhood of a given solution x are all the binary assignments of Hamming distance 1 from x., i.e. with only one bit flipped. The algorithm C_L that produces a better neighboring solutions checks whether flipping any single bit improves the cost and, if not, returns x.

Similar to other TFNP classes a problem is contained in PLS if and only if it can be polynomial time reduced to CircuitFlip. However, the process of reducing to CircuitFlip isn't as simple as reducing to any of the complete problems of the other classes because of complications added by the presence of a specific neighborhood function.

To overcome this, [JPY88] showed that as long as a problem has some local optimization structure, it does in fact belong in PLS, by showing almost all such problems can be reduced to CircuitFlip.

Interesting complete problems in this class are, among other, MaxCut and the travelling salesman problem un the Lin-Kernighan heuristic. [Pap92].

2.2.3 PPA

The third principal TFNP class stands for polynomial parity argument, and is related to the handshaking lemma. PPA problems generally have some sort of pairing property, like for example in the problem of finding a second hamiltonian cycle given one. Its existence is guaranteed by a pairing argument.

More generally the canonical PPA-complete problem asks whether in a graph, with exponentially many vertices each having degree equal to two or one, given a node with a single incident edge, we can find another such node.

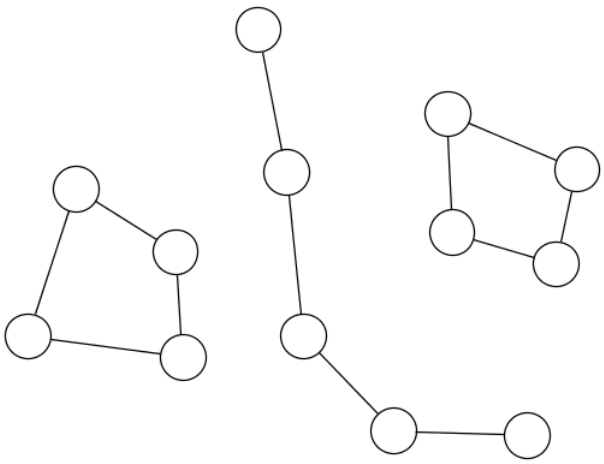

Figure 2.3: Graph of a PPA problem

This problem can be given in circuit form through a circuit that for any input nodes returns either both neighbors or only one, if the degree is equal to 1.

Similar to PPP, this class did not really have any natural complete problems until the work of [FRG17] which demonstrated that ConsensusHalving is in fact PPA-complete.

The reason that the two previously mentioned classes do not have many complete problems is that while they capture important combinatorial existence principles, they were only meant as intermediate classes in capturing a very specific form of solution whose existence is guaranteed, but no efficient algorithm is known.

In particular, at the intersection of PPP and PPA lies PPAD, which contains the computation of a mixed nash equilibrium for two player games.

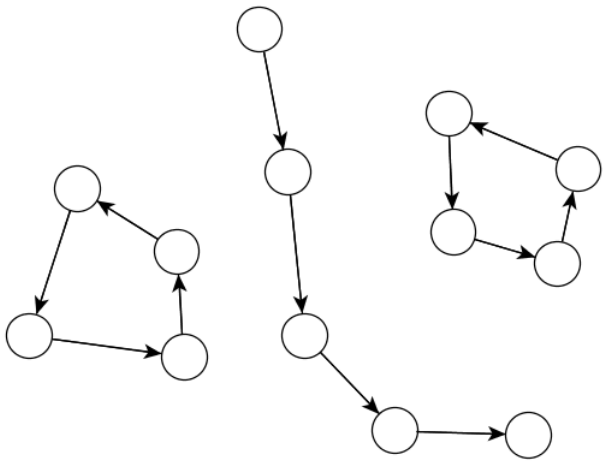

Figure 2.4: Graph of a PPAD problem

2.2.4 PPAD

PPAD (polynomial parity argument directed) is, as the name implies, the directed version of PPA. In particular, now the circuit of the complete problem separately gives two solutions. The first is the predecessor of a node, if it exists, and the second is the successor, if it exists as well. Because the induced graph is now directed we now have sources and sinks, as well as predecessors P and successors S. We say that the induced graph has a directed edge from A to B if and only if P(B)=A and S(A)=B. The fundamental question of PPAD is to ask for a node who either has no predecessor or no successor. This is the EndOfTheLine problem.

Definition 2.2.5. Given two circuits that return for each node a predecessor and a successor, and given a node who's successor or predecessor is itself, find another such node.

Note that unlike PPA we also have directedness here. An interesting property of PPAD is that if we took the naive route and attempted to ask what the node at the other end of the directed path is, then the problem would in fact be PSPACE-hard! [Pap94]. The fact that we ask for any sink or source node in the graph places it far lower in the hierarchy of complexity.

Furthermore, we mentioned earlier that PPAD lies at the intersection of PPP and PPA. This is not apparent from the definition as no mention of collision is made.

Theorem 2.2.1. $PPAD \subset PPP$

[Pap94]. To show this we only have to define a function from the set of the nodes onto itself so that either the element that corresponds to a specific element 0 (here our starting source) or a collision, will permit us to find a sink. Indeed, that function f is defined as $f(A) = S(A)$, if it exists and $f(A) = A$ otherwise. Notice that a collision $f(A) = f(B) = A = S(B) = S(A)$ would mean that A is the required sink. Notice, also, that the PPP oracle can't return $P(0)$ since we labelled as 0 the starting source. ■

Some of the most important problems complete in PPAD are Brouwer and Kakutani [K+41], which are the problems asking for the solutions whose existence is guaranteed by the eponymous theorems. Note that these theorems are connected to this discrete combinatorial class through Sperner's construction.

It is a curious aspect of this class that it seems to accumulate the completeness of so many interesting problems, compared to its more fundamental components PPA and PPP, which only have one or two complete problems.

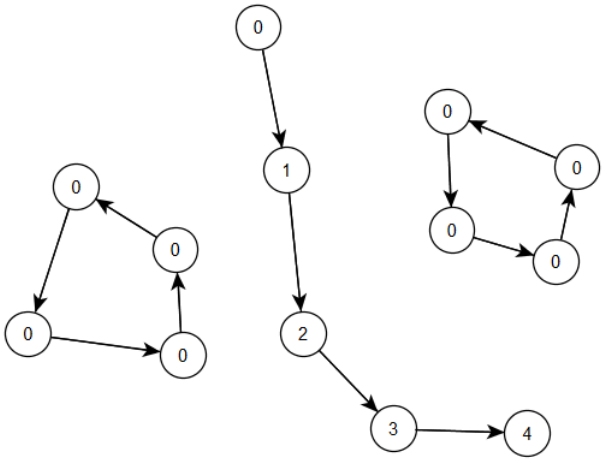

Figure 2.5: Graph of a truthful EndOfTheMeteredLine *problem*

2.2.5 CLS

Last but not least, we mention the class CLS. CLS was introduced by Daskalakis et al in [DP11] in order to capture the difficulty of certain continuous (and hence amenable to PPAD theorems) and optimization (and hence amenable to local search) problems. This class lies at the intersection of both PPAD and PLS. Typical problems in this class include mixed nash equilibria in congestion games (where we have both PLS and PPAD existence arguments) among other.

The first definition for CLS proposed by Daskalakis et al was somewhat artificial, combining both a PLS and PPAD problem and tryign to solve them at the same time. Specifically, their definition of a typical CLS-complete problem was the following.

Definition 2.2.6 (Continuous LocalOpt [DP11]). Given two functions f and p assumed to be Lipschitz continuous, and ϵ,λ >0,find an ϵ-approximate fixpoint of f with respect to p, or two points that violate the λ-continuity of p or of f.

Hence, CLS is a close relative to PLS, in which not only the potential function is continuous, but also the neighborhood function is continuous. It is the continuity of the neighborhood function that also places it inside PPAD; indeed, an approximate fixpoint of the neighborhood function (which can be found within PPAD) is a solution.

The problem with Continuous LocalOpt is that it is not very natural, attempting to merge two different functions in an artificial way.

For this reason, a novel CLS-complete problem was proposed by Hubacek et al [HY17]. This problem is called EndOfTheMeteredLine.

Definition 2.2.7 (EndOfTheMeteredLine [HY17]). Given circuits $S, P : \{0,1\}^n \to \{0,1\}^n$, and $V : \{0,1\}^n \to [2n] \cup \{0\}$ such that $P(0^n) = 0^n \neq S(0^n)$ and $V(0^n) = 1$, find a string $x \in \{0,1\}^n$ satisfying one of the following:

- either $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^n$
- $x \neq 0^n$ and $V(x) = 1$
- either $V(x) > 0$ and $V(S(x))-V(x) \neq 1$ or $V(x) > 1$ and $V(x)-V(P(x)) \neq 1$

Practically, EndOfTheMeteredLine instances look somewhat like Figure 2.5.

Notice that this problem is at least as easy as a PPAD problem, since we also have the additional information of the distance on the nodes. Obviously, however, the truthful behaviour of the V function cannot be guaranteed, which is why, in usual TFNP fashion, we add possible violations as solutions that can be returned.

Chapter 3

Polynomial Local Search

PLS, short for Polynomial Local Search, is the class whose relations embody the difficulty of computing any local optimum in any sort of discrete local optimization setting. In other words, the relation that must be satisfied is a set of constraints that show every neighboring solution has worse value. Defining PLS as a set of total problems instead of relations we have the following definition, given by Johnson, Papadimitriou and Yannakakis in their seminal paper introducing PLS. [JPY88]. For an complete picture of PLS and techniques associated with local search we refer the reader to [AAL03] and [MAK07].

Definition 3.0.1. A problem L in PLS consists of a set of instances D_L, and a set of solutions F_L for each instance, similar to TFNP. Moreover, for each solution s we have a polynomial time function $c_L(s, x) \in \mathbb{N}$, a set $N(s, x) \subset F_L(x)$ called the neighborhood of s as well as the following two algorithms A_L, C_L. Algorithm A_L produces any solution s given an instance x. Algorithm C_L given an instance x and a solution s either returns a solution $s' \in N(s, x)$ with $c_L(s', x) < c_L(s, x)$ or reports that no such solution exists and hence s is locally optimal.

The very definition of PLS suggests an immediate algorithm to compute such a local optimum of a PLS problem.

Algorithm 1: Standard Algorithm

```
s ← A_L(x)
if c_L(A_L(s, x)) < c_L(s, x) then
   s ← A_L(s, x)
else
   return s
end if
```

Indeed, this algorithm will iteratively improve the value of the solution in the simplest way possible until it arrives to a solution that cannot be further improved. Of course, its efficiency is not guaranteed and in most PLS problems it can be shown that it takes exponential time with respect to the size of the description of the problem to run.

The first question one might consider is, given the cost functions and the algorithms that choose the next neighbor each time (i.e. the pivot functions), whether it's possible to obtain the output of the standard algorithm without having to run it in its entirety. In other words, can shortcuts exist for computing the final destination of a specific path along the transition graph? This is called the "Standard Algorithm Problem"

The answer is negative by the following theorem, also proved in the paper introducing the PLS class to the world [JPY88].

Theorem 3.0.1. *There is a PLS problem such that its corresponding Standard Algorithm Problem is NP-hard*

To see this, one only needs to construct a PLS problem that emulates SAT. Specifically we choose the set of solutions to be variable assignments of a SAT formula, with each solution's neighbors being the succeeding and preceding number in its binary representation and the cost being equal to the binary

representation, except if its a satisfying assignment in which case its 0. By beginning from the all true assignment, the standard algorithm would move downwards until it detects a satisfying assignment or reaches 0. In either case, the output solves the satisfiability problem.

Note that the Standard Algorithm Problem is not in TFNP, despite being a search problem, since its solutions cannot be verified succinctly without rerunning the algorithm.

The main interest of PLS problems, however, is not finding a specific local optimum but finding any local optimum, which is no longer NP-hard and hence more tractable and interesting.

3.1 PLS reductions

As with any complexity class, our main tools for giving concrete bounds on the hardness of a problem are reductions. In this section, we seek to present the specific reductions used to establish hardness of PLS problems as well as draw unconditional conclusions regarding the intractability of local searching their solution domain.

As in the theory of NP-hardness, a PLS problem *reduces* to another, if we can transform instances from one problem to another, and through a solver for one problem obtain a solution for the other. Specifically, $A \leq_P LSB$ means that instances of A are transformed into instances of B, and therefore A is "easier" than B.

Definition 3.1.1. [JPY88]

A problem A is PLS-reducible to a problem B if there exist two algorithms f,g such that f maps instances x of A to instances $f(x)$ of B, g maps (solutions of $f(x)$,x) pairs to solutions of x, and if s is a local optimum of B then $g(s, x)$ is a local optimum of x

The main technical contribution of this work is such an involved PLS reduction, presented in chapter 4. To clarify, what one needs to do to obtain a PLS-reduction is provide a consistent way to transform given "hard" arbitrary instances of problem A, into instances of of problem B, so that from a local optimum of the newly created instance we can extract a local optimum of the initial problem. Note that notion of reduction has all the desirable properties of NP-reductions, such as composability.

Before we go on to define PLS-complete problems, however, we present a stronger form of reduction, namely *tight* PLS-reductions that permit transferring more properties than mere PLS-hardness from one problem to the other. To do this we define the transition graph of a PLS-problem.

Definition 3.1.2 ([Sch91]). The transition graph of an instance I of a PLS problem L is a directed graph with one node for each solution and one $s \to t$ edge for each (s, t) pair such that $t \in N(I, s) \wedge c_L(t, I) < c_L(s, I)$

In short, the transition graph shows the possible path a local search algorithm might take. Obviously, since it is acyclic, it possesses at least one sink node, which shows the totality of the problem.

Using this graph we now present an augmented version of a PLS-reduction, which not only allows the transference of the local optima from one instance to the other, but also conserves the structure of the transition graph. If such a reduction with the following properties is achieved, in addition to the reduction mentioned before, we obtain stronger bounds on the runtime of local search.

Definition 3.1.3 ([Sch91]). Let P, Q be PLS problems, and let (f,g) be a PLS reduction from P to Q. We say that the reduction is tight if for any instance I of P we can choose a subset $\mathbb{R}$ feasible solutions for the image instance J=f(I) of Q so that the following properties are satisfied:

- (1) contains all local optima of J.
- (2) For every feasible solution p of I, we can construct in polynomial time a solution q of J such that g(q,I)=p.
- (3) Suppose that the transition graph of J, TG(J) contains a directed path $q \to q$', such that $q, q' \in \mathbb{R}$, but all internal path vertices are outside $\mathbb{R}$, and let p=g(q,I) and p'=g(q',I) be the corresponding feasible solutions of I Then, either $p = p$' or TG(I) contains an arc from p to p'.

To give some intuition for this definition, we have to consider a PLS-reduction as an injective function from the transition graph of problem P to the transition graph of problem Q. As function f sends solutions of the initial instance to certain nodes of the transition graph of the second problem. The crux of the notion of tightness is that the paths between nodes along the transition graph will not be shortened, i.e. the path $A \to B$ is at most as long as $f(A) \to f(B)$.

The reason this is important, is that if we knew that the lengths of the paths of the initial transition graph are long then the new instance constructed has even longer such paths, and hence allows bounds on the performance of any algorithm that is forced to follow the improvement path strictly, to be transferred unconditionally to the new problem.

Specifically, in the same paper [Sch91] it is proven that the archetypical PLS-complete problem CircuitFlip (which we will define rigorously in the next section) has the following properties:

- The Standard Algorithm Problem (SAP) is PSPACE-complete (note how this is even stronger that 3.0.1)
- There are initial solutions for which any local search algorithm (i.e. like SAP) would be forced to visit exponentially many nodes before reaching the sink

These properties, as long as one uses strictly tight PLS-reductions, are immediately transferred to any PLS-complete problem. Note that these properties constitute unconditional lower bounds on the performance of strict local search and would hold even if PLS turns out to be in FP. Since most PLS-reductions today are tight we have these properties for almost all PLS-hard problems. This means that a potential polynomial algorithm for calculating a local optimum is extremely unlikely to be achieved by following strictly the local improvement path. Instead it would have to cut across the solution space, just like the Ellipsoid Method and the Interior point algorithm.

3.2 PLS complete problems

In this section we give an overview of the major problems that have been proven to be PLS-complete, using reductions similar to the previous section. This will illustrate the basic combinatorial nature that underlies all such problems, as well as show that even very natural local optimization problems are PLS-complete.

We begin with the archetypal PLS-complete problem, namely CircuitFlip. This problem holds the same importance with respect to the PLS class as SAT does for NP.

Definition 3.2.1 ([JPY88]). An instance of CircuitFlip is a feedback-free Boolean Circuit made up of AND, OR and NOT gates with n input bits $(x_0, x_1, x_2, ..., x_n)$ and m output bits $(y_0, y_1, y_2, ..., y_m)$. The cost $c_L(x)$ is the binary value associated with its output, i.e. $\sum 2^i * y_i$. The neighborhood of a given solution x are all the binary assignments of Hamming distance 1 from x, i.e. with only one bit flipped. The algorithm C_L that produces a better neighboring solutions checks whether flipping any single bit improves the cost and, if not, returns x.

Note that, while CircuitFlip is a minimization problem, it can easily be formulated as a maximization problem, simply by flipping the output bits. From now on when we use the term "cost" we refer to the minimization version, while when we use the term"value" we refer to the maximization version.

The first thing one must do to show that CircuitFlip does in fact capture all of PLS is to reduce every PLS problem to it. A full proof of this fact is provided in [JPY88]. We give here an overview.

Theorem 3.2.1. CircuitFlip *is PLS-complete.*

Proof Sketch. The main difficulties in proving this theorem lie in adapting the neighborhood structure of an arbitrary PLS problem to CircuitFlip. To this end we have to first reduce to an intermediate problem where the neighborhood can match the local neighborhood structure of the CircuitFlip problem.

Specifically, we reduce to two intermediate problems. First, we restrict each solution to have only one improving neighbor. Second, we have to reduce to a problem where the solution strings have the flip neighborhood as an improvement function. Finally, we add a circuit to calculate the cost function. ■

Therefore, CircuitFlip is at least as hard as any problem that incorporates some local search heuristic combinatorial structure. While CircuitFlip is a useful problem that we can now use to show other problems as PLS-complete, it is unwieldy and rather unnatural. The reason is that circuits are implicit in its description, despite the fact that most problems we encounter in practice make no mention to circuits and boolean logic.

To alleviate this, Schaffer and Yannakakis [Sch91], using an involved reduction from CircuitFlip to POS-NAE3SAT, and then from POS-NAE3SAT to MaxCut showed that MaxCut is PLS-complete under the flip heuristic. This is an extremely natural problem, which allowed a vast amount of other problems to be shown PLS-complete. In addition, their proof was also tight, as in 3.1, which showed that the flip heuristic in a MaxCut instance can not only take an exponential number of steps to finish, but it is also PSPACE-hard to compute its ending state.

To stress its importance, we define MaxCut below.

Definition 3.2.2 (MaxCut [Sch91]). An instance of MaxCut consists of a simple undirected graph G (V, E) with positive weights on the edges. A feasible solution is a partition of V into two sets V1, V2 (not necessarily of equal size). The measure of a solution is the weight of the cut, i.e. the total weight of the edges having one endpoint in each half of the partition, and optimal solutions have maximum measure. Two solutions are neighbors if one can be obtained from the other by moving a single vertex from one side of the partition to the other side.

The combinatorial structure of this problem is in fact so natural and simple, that it is often very easy to find it even in very simple settings, allowing us to bring the complexity intractability bounds of CircuitFlip down to our specific problem.

One of the places that this aforementioned structure can be found is in the local best response improvement dynamics of congestion games. Often, one corresponds the players of a congestion game with the nodes of a MaxCut instance, while encoding the edge weights in the interactions between players, in order to show that the local best response dynamics behave in the exact same way as the flip heuristics of MaxCut. Since the fixed points of best response dynamics, i.e. configurations where no player has any incentive to change, are the Pure Nash Equilibria of a game, such a correspondence shows that finding the PNE of the particular game is PLS-complete.

In Chapter 5 we construct certain reductions of this sort, mostly using MaxCut as a starting point to show PLS-completeness of certain games. However, our work reveals that in certain games, where the interactions between players cannot encode arbitrary edge weights, MaxCut is insufficient for showing PLS-hardness. This leads as to consider a very natural extension of MaxCut, named NodeMaxCut, which is exactly the same as MaxCut only now the graph is node-weighted instead of edge weighted. A direct reduction from MaxCut to NodeMaxCut is in fact unlikely to be achievable since edge weights for every pair of nodes cannot be emulated only with weighted nodes. Because of this, we have to consider a more complicated reduction from CircuitFlip to NodeMaxCut, which shows the fundamental aspect of our problem.

The reductions from CircuitFlip below are by no means an exhaustive list of all PLS reductions. For instance, [DM13] and [DMT09] prove PLS-completeness for their problems in an entirely different way. Here we focus on PLS-reductions in graph theoretic problems.

3.3 Reductions from CircuitFlip

3.3.1 The Schaffer-Yannakakis construction

The most important PLS problem CircuitFlip was historically reduced to, is the archetypal MaxCut problem. The reduction that achieved this in [Sch91] possesses several powerful characteristics that were later reused in further PLS proofs, including our own. In this section we give a concise and non-rigorous presentation of the inner workings on the proof, especially remarking the parts that we intend to reuse in further sections.

3.3.1.1 High level architecture

The construction presented by Schaffer and Yannakakis [Sch91] consists of a POSNAE3SAT instance that emulates a CircuitFlip instance in the following way: whenever we flip a variable in the POSNAE3SAT

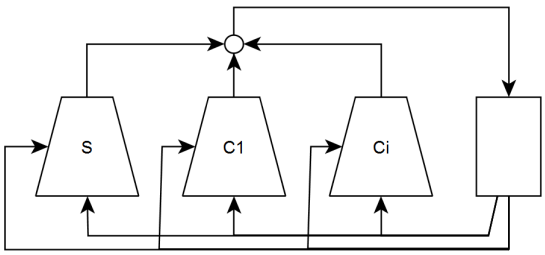

Figure 3.1: The general organization of the gadget used in the [Sch91] proof. Note that the rectangular shape is meant to conceptually represent the gadgets responsible for comparison and unlocking the correct circuit.

instance, it either corresponds to an input of the CircuitFlipinstance being flipped, or to an intermediate step between two such flips. POSNAE3SAT is a variation of classic 3SAT where instead of arbitrary 3SAT clauses we have clauses that are satisfied only if at least one variable has a different truth value from the others. Because a POSNAE3SAT instance can be immediately converted to a MaxCut instance, in the following we will present the proof as if it was directly reducing to a MaxCut instance. Hence, when we use the term "true value" we mean that in the MaxCut instance the node is on the side of the cut corresponding to variables with true value.

> Proof [Sch91] in a nutshell: The MaxCut instance is composed of one central gadget that emulates the main circuit S, as well as n additional such circuits C_i that calculate the values of the neighbor solutions. There are also n nodes D_i, connected in such a way that at most only one has true value at a time. The D_i with the true value is interpreted as the circuit that is meant to provide the improving solution to S. Moreover, there are certain auxiliary gadgets whose purpose is to compare the values of the outputs, incentivise the correct D_i towards the true value, "unlock" the central circuit S so it can take the new solution and finally force all circuits S, C_i to calculate correctly. In an equilibrium state no D_i must have true value, all circuits must be computing correctly, and no C_i must be calculating a value higher than S. This means that MaxCut equilibria are at least as hard as CircuitFlipequilibria, i.e. finding them is PLS-hard.

In order to emulate the entire neighborhood structure the proof needs not only to construct a single circuit-simulating MaxCut gadget, but also to to repeat this construction for each potential neighbor of each solution, as well as implement a way to compare between the outputs of these circuits. That is necessary so that every flip results in a better solution for the CircuitFlip problem.

At the core of this proof lies a very important construction, that is reused by all subsequent PLS reductions from CircuitFlip. This gadget's purpose, which we call the SY (Schaffer-Yannakakis) gadget, is to emulate a NOR gate, with some additional stipulations. By combining these gadgets Schaffer and Yannakakis obtain their S, C_i circuits.

Because of the central importance of the way this gadget functions we present here a detailed overview of its inner workings.

3.3.1.2 The Schaffer-Yannakakis gadget

The construction Schäffer and Yannakakis used to prove MaxCut is PLS-complete consists of certain input nodes, certain output nodes and certain internal nodes. In particular, each such NOR gate is composed of the nodes in the following figure.

Note that the interesting part here is that while we gave the construction in node weighted form, it performs exactly in the same way even if we use weighted edges with $w_{ij} = w_i * w_j$. This is because in the original construction by Schaffer and Yannakakis it was first used to prove the P-completeness of the unweighted case of local MaxCut.

What does this gadget do? It has the following properties: if the control nodes y, z have certain values then the gadget either computes the NOR value of its inputs or "unlocks" its input nodes so that

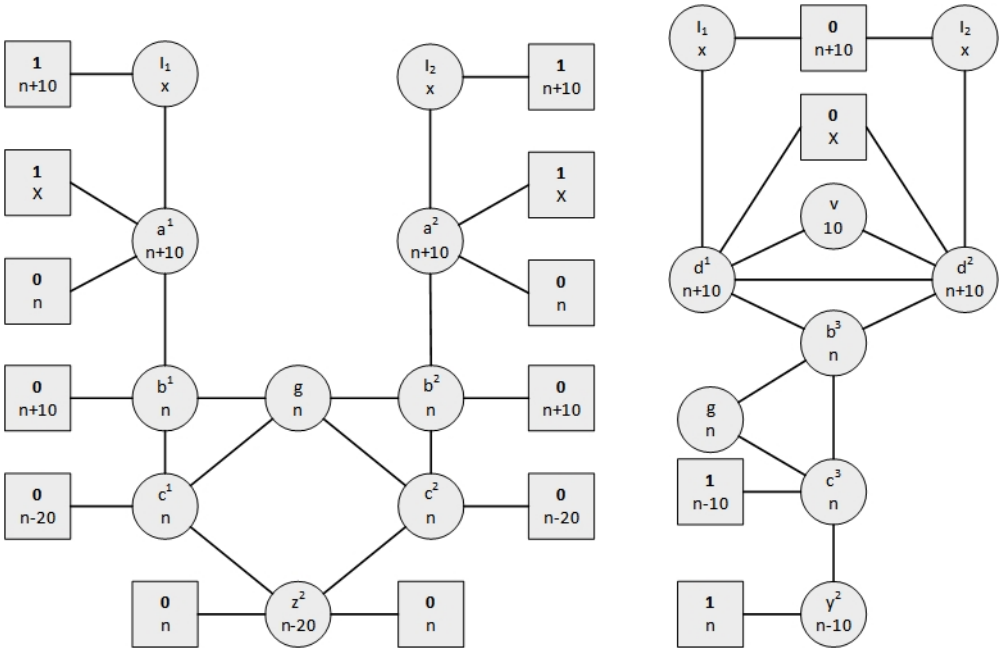

Figure 3.2: The NodeMaxCut *instance implementing a NOR(n) gadget.*

they experience no bias with respect to the gadget.

More specifically, we have the following lemmas.

Lemma 3.3.1. *In an equilibrium, if* $z_i^1 = 1$ *and* $y_i^1 = 0$*, then* $I_1(g_i)$*,* $I_2(g_i)$ *are indifferent with respect to the gadget* G_i*.*

Lemma 3.3.2. *If gate* G_i *is incorrect, then* $z_i^2 = 1$*. If* $y_i^2 = 0$ *then* $z_i^2 = 1$*. If* $z_i^2 = 1$*, then for all* $j < i$ $z_j^1 = z_j^2 = z_j^3 = 1$ *and* $y_j^1 = y_j^2 = y_j^3 = 0$*.*

Lemma 3.3.3. *Suppose* $z_i^1 = 0$ *and* $y_i^1 = 1$*. If* g_i *is correct then* z^2 *and* y^2 *are indifferent with respect to the other nodes of the gate* G_i*. If* g_i *is incorrect then* g_i *is indifferent with respect to the other nodes of the gate* G_i*, but gains the node* ρ *of weight* 2^{-500N}*.*

Lemma 3.3.4. *If Control* $= 1$ *then all* y, z *nodes have a* 2^{-87N} *bias towards their natural values. If Control* $= 0$ *then all* y, z *nodes have a* 2^{-87N} *bias towards their unnatural values.*

Lemma 3.3.5. *Assuming all nodes of the computing circuit gadget are in equilibrium and have no external biases. If Control* $= 1$ *then* $\forall i, z_i^1 = 0, y_i^1 = 1, z_i^2 = 0, y_i^2 = 1, z_i^3 = 0, y_i^3 = 1$*. If Control* $= 0$ *then* $\forall i z_i^1 = 1, y_i^1 = 0, z_i^2 = 1, y_i^2 = 0, z_i^3 = 1, y_i^3 = 0$*.*

What one should particularly note is that these control nodes have extremely low bias to either direction, despite the fact that they define the behaviour of the entire construction based on their value. Indeed, that is the fundamental reason the reduction itself functions. We stress the importance of these NOR gadgets as they are reused practically intact in our reduction in the next chapter.

3.3.2 The simplified two circuit construction

However, the framework utilized by Schaffer and Yannakakis is somewhat unwieldy when one needs to adapt it for further PLS-completeness proofs from similar graph theoretic problems. In these cases, one can use certain simpler PLS-completeness reduction gadgets proposed by [ET11].

Two such cases are the following. The first is the PLS-hardness of MaxCut with the added stipulation that instance graphs have maximum degree 5 and OneEnemyPartyAffiliation, which is a variation of MaxCut only with all except one edge for each node being negative. Because of their significant

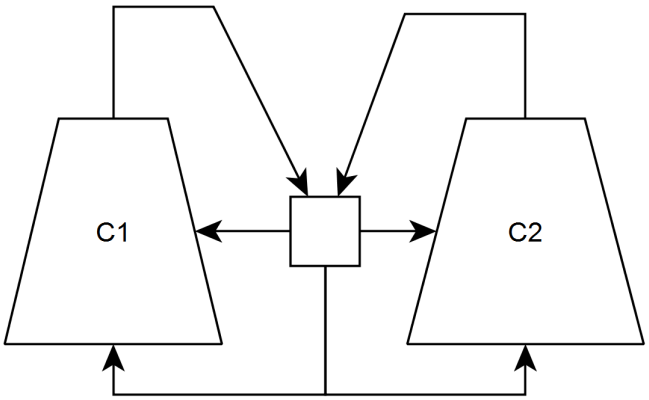

Figure 3.3: The simplified flip-flop architecture.

similarity to MaxCut, Elsasser and Tscheuschner modified and simplified the original proof of Schaffer and Yannakakis to use it on their more complex problem.

The principal modification that both of these approaches adopt is that instead of having N circuits calculating the value of each neighbor and then comparing to choose the improving flip, Elsasser and Tscheuschner use only two circuits that each output an improving solution if one exists, along with the value of the current solution. This greatly simplifies things, considering that we now dont have to bother with comparing multiple values among other details. This two-circuit approach was also used by Gairing and Savani to prove PLS-completeness [GS10] using the Schaffer-Yannakakis gadgets.

Note that the reductions using only two circuits themselves borrow from an even older construction by Krentel [Kre89], showing PLS-completeness for certain other local optimization problems. Furthermore, this two circuit construction has the unfortunate property that the reduction is no longer tight. This is due to the fact that while local optima of the construction still correspond to local optima of CircuitFlip, each improving flip might flip multiple bits at the same time, hence creating shortcuts in the process of local search. Regardless, in this work we are concentrated on the computational hardness of the local optima themselves, rather than, albeit unconditional, guarantees on the performance of local search itself.

Chapter 4

Node Weighted Max Cut

The main technical work of this thesis is presented in this section. In particular, this chapter contains the full details of our proof reducing CircuitFlip to NodeMaxCut. We remind that the proof in this chapter heavily draws from the proofs of [ET11] and [GS10] which in turn draw from [Sch91]. Firstly, we present the problem and give an intuitive motivation for its formulation. Next, we explain why a direct reduction from MaxCut to NodeMaxCut is unlikely to exist and why it is necessary to reduce from the much more fundamental CircuitFlip problem. Secondly, we present the overview of our reduction, along with proofs for the behaviour of each gadget.

4.1 The problem

The central problem we are examining in this chapter comes from a slight simplification of MaxCut. Specifically, now we assume nodes each have their own weight and each node, instead of choosing a partition such that $\sum_{i,j \in V_x} w_{ij}$ is minimized, the node minimizes $\sum_{i,j \in V_x \cap N(i)} w_j$. In NodeMaxCut the node strives to choose a side with as light neighbors as possible. Note that an immediate combinatorial difference is that now every node "sees" each other the same way, i.e. a very heavy node greatly influences all his neighbors, as opposed to vanilla MaxCut where he might have heavy and light adjacent edges, influencing his neighbors non-uniformly.

Definition 4.1.1 (NodeMaxCut). An instance of NodeMaxCut consists of a simple undirected graph G (V, E) with positive weights (W_i) on the nodes and unweighted edges. A feasible solution is a partition of V into two sets V1, V2 (not necessarily of equal size). The measure of a solution is the weight of the cut, where the edges e_{ij} are considered as if having weight $W_i * W_j$. Optimal solutions have maximum measure. Two solutions are neighbors if one can be obtained from the other by moving a single vertex from one side of the partition to the other side.

NodeMaxCut instances are essentially MaxCut instances, where one asks the question: what if we only had node weights? In other words, similar to MaxCut where players seek to maximize the sum of their adjacent edges in the cut, now in NodeMaxCut players seek to maximize the sum of their neighboring vertices on different sides of the cut as them. It is a natural variation of the basic MaxCut. The fact that such a node weighted game does in fact possess a Nash Equilibrium is not trivial. Indeed, one needs to observe that we now have a *weighted* potential function, rather than normal potential function. Specifically, in MaxCut the potential function (i.e. the cut) has the following form.

$$\sum \sigma(i) * \sigma(j) * W_{ij}$$

Where $\sigma(i) \in -1, 1$ depending on the side of the cut.

On the contrary, a player A in NodeMaxCut seeks to maximize

$$\sum_{B \in N(A)} \sigma(A) * \sigma(B) * W_B$$

One immediately notes the necessity for a weight factor for the potential so that it can become exact,

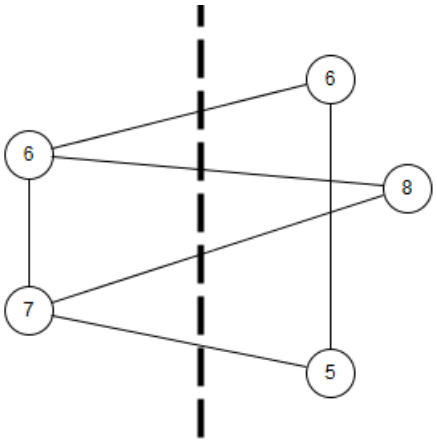

Figure 4.1: An instance of NodeMaxCut. *Note how node weights are equivalent to $W_i * W_j$ edges, in terms of each player's local dynamics*

making the problem equivalent to having a graph with edges of the form $W_i * W_j$.

$$\sum \sigma(i) * \sigma(j) * W_i * W_j$$

The main setting NodeMaxCut instances arise in, to the best of our experience so far, is Weighted Congestion Games with Identity Delays. In these games we cannot conceal edge weights w_{ij} in the delay functions so as to reduce from MaxCut. In fact such Congestion Games are most common in the real world as opposed to adversarially constructed delay functions. Indeed when we use modern roads they each have similar susceptibility to congestion instead of exponentially different coefficients. This similarity between resources means that we need a different problem to capture their PLS-completeness, i.e. to obtain PLS-hardness solely through the exponentially different weights of players. For this reason, we define our NodeMaxCut problem and examine its local search complexity.

4.1.1 A naive unsuccessful approach

The first question one asks is whether PLS-hardness for NodeMaxCut can be immediately obtained through a reduction from the usual MaxCut problem, obviating the need for such a cumbersome reduction. To see why this is unlikely to be true, let us consider the simplest possible version of such a reduction. In particular, such a reduction would consist of a MaxCut instance where certain nodes correspond to the nodes of the NodeMaxCut instance, in a bijective manner.

Since the neighborhood structures in these two constructions must match for the PLS reduction to be valid, a flip of a node in the red graph must correspond to a flip of a node of the blue graph. Moreover, suppose in the red MaxCut instance the node being flipped gains $\Delta = \sum W_{ij}$ by being flipped. Since the correspondence must be a bijection, flipping the node in the blue graph also gains $\Delta = \sum W_{ij}$. Now consider what happens to value of the cut (i.e. potential) after such a flip. At first, it changes by Δ. Since the other dark blue nodes must somehow be "informed" of the fact that this node was flipped, certain light blue nodes must flip, solely through local improvement. However, their local improvement flips also strictly increase the value of the cut and hence the value of the cut is now $\Delta + \delta_1$. Now if we flip the deep blue node we initially flipped back to its original side we must decrease the cut again exactly by Δ, since the other dark blue nodes never changed. Afterwards, the light blue nodes again flip in order to inform the other dark blue nodes of the change of their neighbor. This again increases the cut by δ_2 to a total of $\delta_1 + \delta_2$. Notice that both the starting and end states of this process correspond to the exact same partitions of the nodes of the original red MaxCut graph. Therefore, by finiteness, there must be a configuration of the blue NodeMaxCut graph such that both values of the cut are the same. However this would imply that $\delta_1 = \delta_2 = 0$ which can only happen if no light blue nodes move after a flip of the "core" dark blue nodes has been achieved. This is impossible since the neighborhood of the dark blue nodes must change whenever one of them changes (in order to emulate MaxCut) and this cannot be achieved with constant light blue nodes in a NodeMaxCut instance. In

the above reasoning one could replace the dark blue nodes with sets of nodes, and yet they would run into the same problem: as long as intermediate nodes are used, they will absorb a nonzero part of the improvement of every move, making a one-to-one correspondence with a MaxCut instance impossible to achieve. The above difficulties strongly suggest that NodeMaxCut is in fact a more fundamental problem whose PLS-hardness cannot be derived directly from MaxCut. For this reason we turn to the "grandmother" problem of the PLS complexity class, namely CircuitFlip.

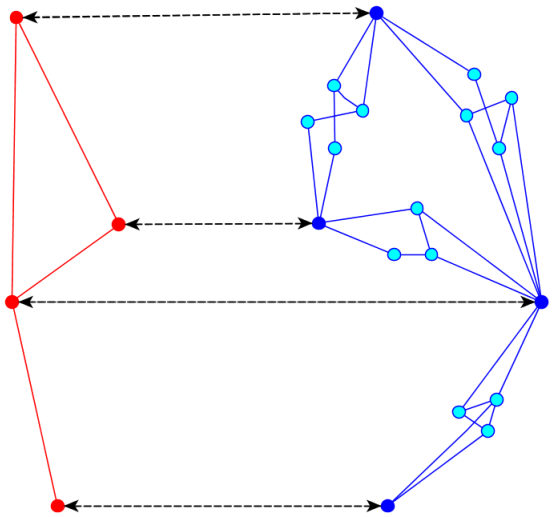

Figure 4.2: How a naive reduction from NodeMaxCut *would have to work. Initial* MaxCut *instance is in red, the deep blue nodes are the corresponding nodes of* NodeMaxCut*, and light blue are any nodes that the reduction would add to emulate* MaxCut

Definition 4.1.2. An instance of CircuitFlipcan be described as a feedback-free Boolean Circuit made up of AND, OR and NOT gates with n input bits $(x_0, x_1, x_2...x_n)$and m output bits $(y_0, y_1, y_2...y_m)$. The cost $c_L(x)$ is the binary value associated with its output, i.e. $\sum 2^i * y_i$. The neighborhood of a given solution x are all the binary assignments of Hamming distance 1 from x., i.e. with only one bit flipped. The algorithm C_L that produces a better neighboring solutions checks whether flipping any single bit improves the cost and, if not, returns x.

Of course, NodeMaxCut is not the first problem that requires reducing from the oldest known PLS problem. Recall the two problems presented in the previous section, which utilized a *flip-flop architecture* to reduce from CircuitFlip. Our proof draws a significant amount of ideas from these proofs. The differences are highlighted in the next section.

4.2 Overview of the reduction

In our proof, we slightly simplify certain details of the two previous reductions to obtain an even simpler construction, which combined with our insights into the nature of NodeMaxCut and the gadgets that this entails, allows us to obtain a PLS-hardness proof for our problem. More specifically, we use the two-circuit construction of Elsasser and Tscheuschner, with the added simplification that now instead of two central nodes controlling which of the two circuits is active, we have only one such node (the active circuit depends on the value of that node) connected in a somewhat involved way with the outputs of the circuits so that we can achieve comparison. This is necessary because of the combinatorial structure of NodeMaxCut as opposed to all preceding problems where edges have weights rather than nodes, which would cause significant difficulties if we were to directly use the preexisting reductions. Furthermore, and more importantly, we create a gadget that allows nodes with small weights to influence nodes with heavy weights, which is the key observation into the nature of NodeMaxCut as a weighted potential game that allows the reduction to work. In short, it exploits the fact that the weights of nodes enter into

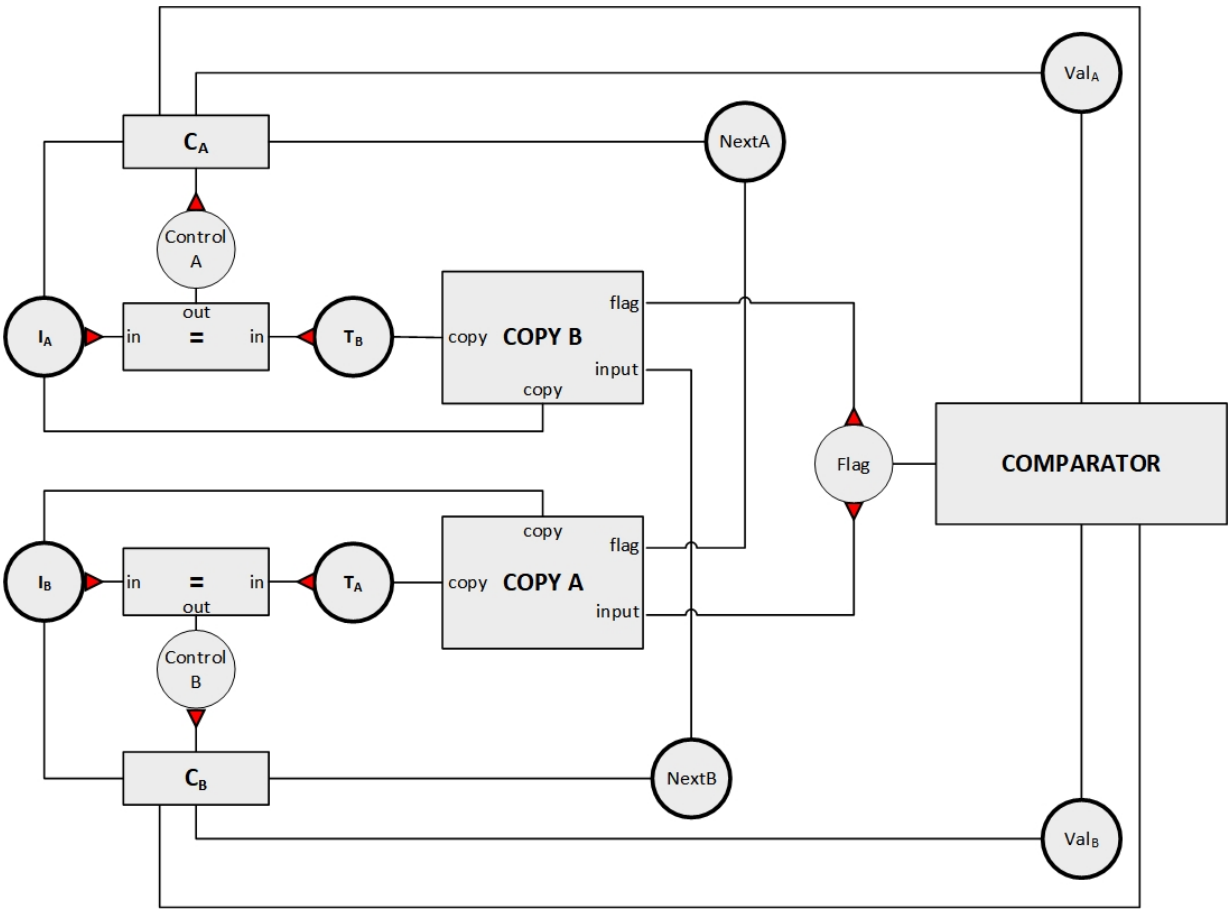

Figure 4.3: The high level construction of the NodeMaxCut *instance. Note that rectangles represent gadgets that will be defined below, circles represent nodes that take part in multiple gadgets, and bolded black circles represent a set of (n) such nodes. The computation gadgets are represented by C_A and C_B.*

the differential of the potential as a coefficient in weighted games, and hence allows very small reductions in the potential to happen in several improving steps. We are now going to describe our reduction in detail along with proofs of the lemmas that dictate the behaviour of the construction. The only lemmas that will not be proved are the ones in Section 3.3.1.2 because they are identical to the ones provided by Schaffer and Yannakakis. We remind that we are reducing CircuitFlipto NodeMaxCut. This means we are constructing an instance of NodeMaxCut that emulates a pair of circuits giving their outputs as inputs to each other. Given a circuit C of CircuitFlip, the node-weighted graph that we construct is a combination of different gadgets which themselves might be seen as smaller instances of NodeMaxCut . These gadgets have different goals but what each of them intuitively does is receiving some information to some of its nodes, the "input" nodes, and transforming it while carrying it through its internal part towards the "output" nodes. These input and output nodes are the nodes through which the different gadgets are connected. The connections are not always simple and we will later go into more detail on how these connections are done.

As before, our construction follows a *flip-flop architecture* that has been previously used for reductions from CircuitFlip to MaxCut and some of its variants, but requires more sophisticated implementations in many of its gadgets, since we deal with the special case of NodeMaxCut. The most important differences and our major technical contributions are summarized at the end of the section. The constructed instance is presented at a high level in Figure 4.3. Next, we discuss the role of each separate gadget. The exact construction of each separate gadget is presented in its respective section.

For a given circuit C of the CircuitFlip with n input gates and m output gates, we first construct the *Circuit Computing* gadgets $C\ell$ (for $\ell = A, B$). These gadgets are instances of NodeMaxCut that simulate C in the following sense: I_ℓ is a set of n nodes whose values correspond to an "input string"

of C. Val_ℓ is a set of m nodes whose values correspond to the output of circuit C with input string I_ℓ. $Next\ell$ is a set of n nodes that represent a neighbor string of I_ℓ with greater output value in C. More precisely, if the "input" string defined by the values of I_ℓ, has a neighbor solution (Hamming distance *1*) with *strictly* greater cost then the values of the n nodes in $Next\ell$ correspond to this neighbor string. In case such a neighbor string does not exist (which means that I_ℓ is an optimal solution to CircuitFlip) the values of the nodes in $Next\ell$ equal I_ℓ.

These *Circuit Computing* gadgets have two separate functionalities: the *write mode* ($Control_\ell = 0$) and the *compute mode* ($Control_\ell = 1$). When C_ℓ is in the *write mode* the values of the input nodes I_ℓ are changed. When C_ℓ is in the *compute mode* the values of the nodes $Next\ell, Val_\ell$ are updated with the correct output values of the circuit C. To formalize the term *correct*, we introduce the following notation that we use through the section.

Definition 4.2.1. For the given circuit C of CircuitFlip. We denote with:

- *Real-Val*(I_ℓ) the value of the circuit C with input string defined by the values of the nodes in I_ℓ.
- *Real-Next*$(I)_\ell)$ is a neighbor string (Hamming distance *1*) of I_ℓ with strictly greater output value in circuit C. If such a string does not exist, *Real-Next*$(I_\ell) = I_\ell$.

The *Comparator* gadget compares Val_A and Val_B, which are intended to be *Real-Val*(I_A), *Real-Val*(I_B) and outputs *1* if $Val_A \leq Val_B$ or *0* otherwise. The result of this comparison is "*stored*" in the value of the *Flag* node. If $Flag = 1$ then I_B "*writes*" her better neighboring solution to I_A (symmetrically if $Flag = 0$). Intuitively, if this happens then in the next "*cycle*" I_A will have a better value and will write her improving solution to I_B. This goes on and on until no better neighbor solution exists and both I_A and I_B have the same output node values.

The *CopyB* gadget (respectively for *CopyA*) is responsible for writing the values of the nodes *NextB* to the nodes I_A when $Flag = 1$ (when $Val_A \leq Val_B$). When $Flag = 1$ the nodes T_B take the values of the nodes in *NextB*. Now if $I_A \neq NextB$ then the *Equality* gadget turns the value of the *ControlA* to 0 because $I_A \neq T_B$. Thus C_A enters *write mode* and the nodes in I_A adopt the values of the *NextB* nodes. Then *ControlA* becomes 1 since $I_A = T_B$ and C_A enters "*compute mode*". This means that values *Real-Next*(I_A), *Real-Val*(I_A) are written to the *output nodes NextA*, Val_A.

Before proceeding we present a proof-sketch of our reduction. The mathematically rigorous version of this proof is presented in the proof of Theorem 4.2.11 at the end of the section. We will prove that at any equilibrium of the instance of NodeMaxCut of Figure 4.3 in which $Flag = 1$ (symmetrically if $Flag = 0$) three things hold.

1. $I_A = NextB$
2. $NextB =$ *Real-Next*(I_B)
3. *Real-Val*$(I_B) \geq$ *Real-Val*(I_A)

Once these claims are established we can be sure that the string defined by the values of nodes in I_B defines a locally optimal solution for CircuitFlip. This is because the above 3 claims directly imply that *Real-Val*$(I_B) \geq$ *Real-Val* (*Real-Next* (I_B)) which means that there is no neighboring solution of I_B with strictly greater cost. Obviously we establish symmetrically the above claims when $Flag = 0$.

Since our construction in Figure 4.3 is an instance of NodeMaxCut and the bitwise complement of an equilibrium is also equilibrium the term $Flag = 1$ seems meaningless. In the construction of the *Circuit Computing* gadget (and in the constructions of all the presented gadgets) there also exist two *supernodes*, a 1-node and a 0-node, with huge weight that share an edge. As a result, at any equilibrium these nodes have opposite values. The term $Flag = 1$ means that the *Flag* node has the same value with the 1-node. This notation is also used in subsequent lemmas and always admits the same interpretation. As one can see in the appendix, the construction of the gadgets assume nodes with values always 0 or 1. This can be easily established by connecting one such node with its complementary *supernode*.

In the rest of the section we present the necessary lemmas to make the above presented proof sketch rigorous. To do so, we follow a three step approach. We first present the exact behavior of the *Circuit Computing* gadgets C_A, C_B. We then reason why $I_A = NextB$, which we refer to as the *Feedback problem*. Finally we establish the last two claims which we refer to as *Correctness of the Outputs*.

Circuit-computing gadgets

The *Circuit Computing* gadgets C_A, C_B are the basic primitives of our reduction and are based on the

gadgets introduced by Schäffer and Yannakakis to establish PLS-completeness of MaxCut. This type of gadgets can be constructed so as to simulate any boolean circuit C. The most important nodes are those corresponding to the input and the output of the simulated circuit C and are denoted as I, O. The other important node is the *Control* that switches between the *write* and the *compute mode* of the gadget. Figure 4.4 is an abstract depiction of this type of gadgets. The properties of the gadget are described in Theorem 4.2.1. Its proof is presented in further sections, where the exact construction of the gadget is presented. We first introduce some convenient notation that will help us throughout the proof of completeness.

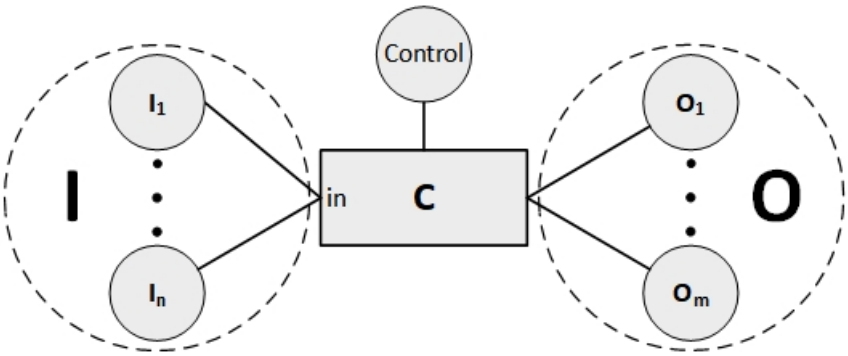

Figure 4.4: Circuit Computing *gadgets. The big, dashed "vertices" named I and O, represent all the input and output nodes respectively. This type of ("hyper"-)node is represented in the rest of the figures with a bold border.*

Definition 4.2.2. Let an instance of NodeMaxCut and a specific equilibrium of this instance. The *bias* that node i experiences with respect to the subset $N' \subseteq N$ is

$$\left| \sum_{j \in N_i^1 \cap N'} w_j - \sum_{j \in N_i^0 \cap N'} w_j \right|$$

where N_i^0 is the set of neighbors of node i that choose 0 (respectively for N_i^1)

Bias is a key notion in the subsequent analysis. The gadgets presented in Figure 4.3 are subset of nodes of the overall instance. Each gadget is composed by the "input nodes", the internal nodes and the "output nodes". Moreover as we have seen each gadget stands for a "circuit" with some specific functionality (computing, comparing, copying e.t.c.). Each gadget is specifically constructed so as at any equilibrium of the overall instance, the output nodes of the gadget experience some bias towards some values that depend on the values of the input nodes of the gadget. Since the output nodes of a gadget may also participate as input nodes at some other gadgets, it is important to quantify the bias of each gadget in order to prove consistency in our instance. Ideally, we would like to prove that at any equilibrium the bias that a node experiences from a gadget in which it is an output node, is greater than the sum of the biases of the gadgets in which it participates as input node.

Theorem 4.2.1 describes the equilibrium behavior of the input nodes I_ℓ and output nodes $Next\ell, Val_\ell$ of the $CircuitComputing$ gadgets C_ℓ.

Theorem 4.2.1. *At any equilibrium of the* NodeMaxCut *of Figure 4.3.*

1. *If* $Control\ell = 1$ *and the nodes of* $Next\ell$,Val_ℓ *experience* 0 *bias from any other gadget beyond* C_ℓ *then:*
   - $\text{Next}\ell = \text{Real-Next}(I_\ell)$
   - $\text{Val}_\ell = \text{Real-Val}(I_\ell)$
2. *If* $Control\ell = 0$ *then each node in* I_ℓ *experiences* 0 *bias from the internal nodes of* C_ℓ.
3. $Control\ell$ *experiences* $w_{Control\ell}$ *bias from the internal nodes of* C_ℓ.

Case 1 of Theorem 4.2.1 describes the *compute mode* of the *Circuit Computing* gadgets. At any equilibrium with $ControlA = 1$, and with the output nodes of C_A being indifferent with respect to other gadgets, then C_A computes its output correctly. Note that because the nodes in $NextA, ValA$ are also connected with internal nodes of other gadgets (*CopyA* and *Comparator* gadgets) that may create bias towards the opposite value, the second condition is indispensable. Case 2 of Theorem 4.2.1 describes the *write mode*. If at an equilibrium $ControlA = 0$ then the nodes in I_A have 0 bias from the C_A gadget and as a result their value is determined by the biases of the *CopyB* gadget and the *Equality* gadget. Case 3 of Theorem 4.2.1 describes the minimum bias that the equality gadget must pose to the Control nodes so as to make the computing gadget flip from one mode to the other. As we shall see, the weights $w_{ControlA} = w_{ControlB} = w_{Control}$ are selected much smaller than the bias the $Control\ell$ nodes experience due to the *Equality* gadgets, meaning that the *Equality* gadgets control the *write mode* and the *compute mode* of the *Circuit Computing* gadgets no matter the values of the nodes in C_ℓ gadgets.

We remark that our construction of the *Circuit Computing* gadgets presented in further sections ensures Theorem 4.2.1 for selecting $w_{Control}$ arbitrarily smaller than the weights for the internal nodes of the *Circuit Computing* gadgets. As we shall see up next this is crucial for our reduction and, as we discuss in the end of the section, this is a major difference with the respective *Circuit Computing* gadgets of other reductions in this vein. This is because of the inclusion of the leveraging gadget that the combinatorial structure of NodeMaxCut dictates.

Solving the Feedback problem.
The purpose of this section is to establish the first case of the above presented claims i.e. at any equilibrium of NodeMaxCut instance of Figure 4.3 in which $Flag = 1$, *NextB* is written to I_A and vice versa when $Flag = 0$. This is formally stated in Theorem 4.2.2.

Theorem 4.2.2. *Let an equilibrium of the instance of* NodeMaxCut *described in Figure 4.3.*

- *If* Flag $= 1$ *then* $I_A =$ NextB
- *If* Flag $= 0$ *then* $I_B =$ NextA

We next present the necessary lemmas for proving Theorem 4.2.2.

Lemma 4.2.3. *Let an equilibrium of the overall* LocalNodeMaxCut *instance of Figure 4.3. Then*

- $ControlA = (I_A = T_B)$
- $ControlB = (I_B = T_A)$

Below we present the construction of the equality gadget. This gadget is specifically designed so that at any equilibrium, its internal nodes create bias to $ControlA$ towards the value of the predicate $(I_A = T_B)$. Notice that if we multiply all the internal nodes of the equality gadget with a positive constant, the bias $ControlA$ experiences towards value $(I_A = T_B)$ is multiplied by the same constant (see Definition 4.2.2). Lemma 4.2.3 is established by multiplying these weights with a sufficiently large constant so as to make this bias larger than $w_{ControlA}$. We remind that by Theorem 4.2.1, the bias that $ControlA$ experiences from C_A is $w_{ControlA}$. As a result, the equilibrium value of $ControlA$ is $(I_A = NextB)$ no matter the values of $ControlA$'s neighbors in the C_1 gadget. The *red mark* between $ControlA$ and the C_A gadget in Figure 4.3 denotes the "indifference" of $ControlA$ towards the values of the C_A gadget (respectively for $ControlB$).

In the high level description of the NodeMaxCut instance of Figure 4.3, when $Flag = 1$ the values of *NextB* is copied to I_A as follows: At first T_B takes the value of *NextB*. If $I_A \neq T_B$ then $ControlA = 0$ and the C_A gadget switches to *write mode*. Then the nodes in I_A takes the values of the nodes in *NextB*. This is formally stated in Lemma 4.2.4.

Lemma 4.2.4. *At any equilibrium point of the* NodeMaxCut *instance of Figure 4.3:*

- *If* Flag $= 1$, *i.e.* NextB writes *on* I_A, *then*
  1. $T_B =$ NextB
  2. *If* $ControlA = 0$ *then* $I_A = T_B =$ NextB

- *If* Flag $= 0$ *i.e.* NextA writes *on* I_B*, then*
  1. $T_A =$ NextA
  2. *If* $ControlB = 0$ *then* $I_B = T_A =$ NextA

In further sections, we present the construction of the *Copy* gadgets. At an equilibrium where $Flag = 1$, this gadget creates bias to the nodes in I_A, T_B nodes towards adopting the values of *NextB*. Since I_A, T_B also participate in the *Equality* gadget in order to establish Lemma 4.2.4 we want to make the bias of the *CopyB* gadget larger than the bias of the *Equality* gadget. This is done by again by multiplying the weights of the internal nodes of *CopyB* with a sufficiently large constant. The "indifference" of the nodes in I_A, T_B with respect to the values of the internal nodes of the *Equality* gadget is denoted in Figure 4.3 by the *red marks* between the nodes in I_A, T_B and the *Equality* gadget.

In Case 2 of Lemma 4.2.4 the additional condition $ControlA = 0$ is necessary to ensure that $I_A = NextB$. The reason is that the bias of the *Copy* gadget to the nodes in I_A is sufficiently larger than the bias of the *Equality* gadget to the nodes in I_A, but not necessarily to the bias of the C_A gadget. The condition $ControlA = 0$ ensures 0 bias of the C_A gadget to the nodes I_A, by Theorem 4.2.1. As a result the values of the nodes in I_A are determined by the values of their neighbors in the *CopyB* gadget.

Proof of Theorem 4.2.2. Let an equilibrium in which $Flag = 1$. Let us assume that $I_A \neq NextB$. By Case 1 of Lemma 4.2.4, $T_B = NextB$. As a result, $I_A \neq T_B$, implying that $ControlA = 0$ (Lemma 4.2.3). Now, by Case 2 of Lemma 4.2.4 we have that $I_A = NextB$, which is a contradiction. The exact same analysis holds when $Flag = 0$. ■

Correctness of the Output Nodes.

In the previous section we discussed how the *Feedback problem* ($I_A = NextB$ when $Flag = 1$) is solved in our reduction. We now exhibit how the two last cases of our initial claim are established.

Theorem 4.2.5. *At any equilibrium of the instance of* LocalNodeMaxCut *of Figure 4.3:*

- *If* Flag $= 1$
  1. Real-Val$(I_A) \leq$ Real-Val(I_B)
  2. NextB $=$ Real-Next(I_B)
- *If* Flag $= 0$
  1. Real-Val$(I_B) \leq$ Real-Val(I_A)
  2. NextA $=$ Real-Next(I_A)

At first we briefly explain the difficulties in establishing Theorem 4.2.5. In the following discussion we assume that $Flag = 1$, since everything we mention holds symmetrically for $Flag = 0$. Observe that if $Flag = 1$ we know nothing about the value of $ControlB$ and as a result we cannot guarantee that $NextB = Real\text{-}Next(I_B)$ or $Val_B = Real\text{-}Val(I_B)$. But even in the case of C_A where $ControlA = 1$ due to Theorem 4.2.2, the correctness of the nodes in $NextA$ or Val_A cannot be guaranteed. The reason is that in order to apply Theorem 4.2.1, $NextA$ and Val_A should experience 0 bias with respect to any other gadget they are connected to. But at an equilibrium, these nodes may select their values according to the values of their *heavily weighted neighbors* in the *CopyA* and the *Comparator* gadget.

The correctness of the values of the output nodes, i.e. $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_A)$, is ensured by the design of the *CopyA* and the *Comparator* gadgets. Apart from their primary role these gadgets are specifically designed to cause 0 bias to the output nodes of the *Circuit Computing* gadget to which the better neighbor solution is written. In other words at any equilibrium in which $Flag = 1$ and any node in C_A: the total weight of its neighbors (belonging in the *CopyA* or the *Comparator* gadget) with value 1 equals the total weight of its neighbors (belonging in the *CopyA* or the *Comparator* gadget) with value 0.

The latter fact is denoted by the *green marks* in Figure 4.5 and permits the application of Case 1 of Theorem 4.2.1. Lemma 4.2.6 and 4.2.7 formally state these "green marks".

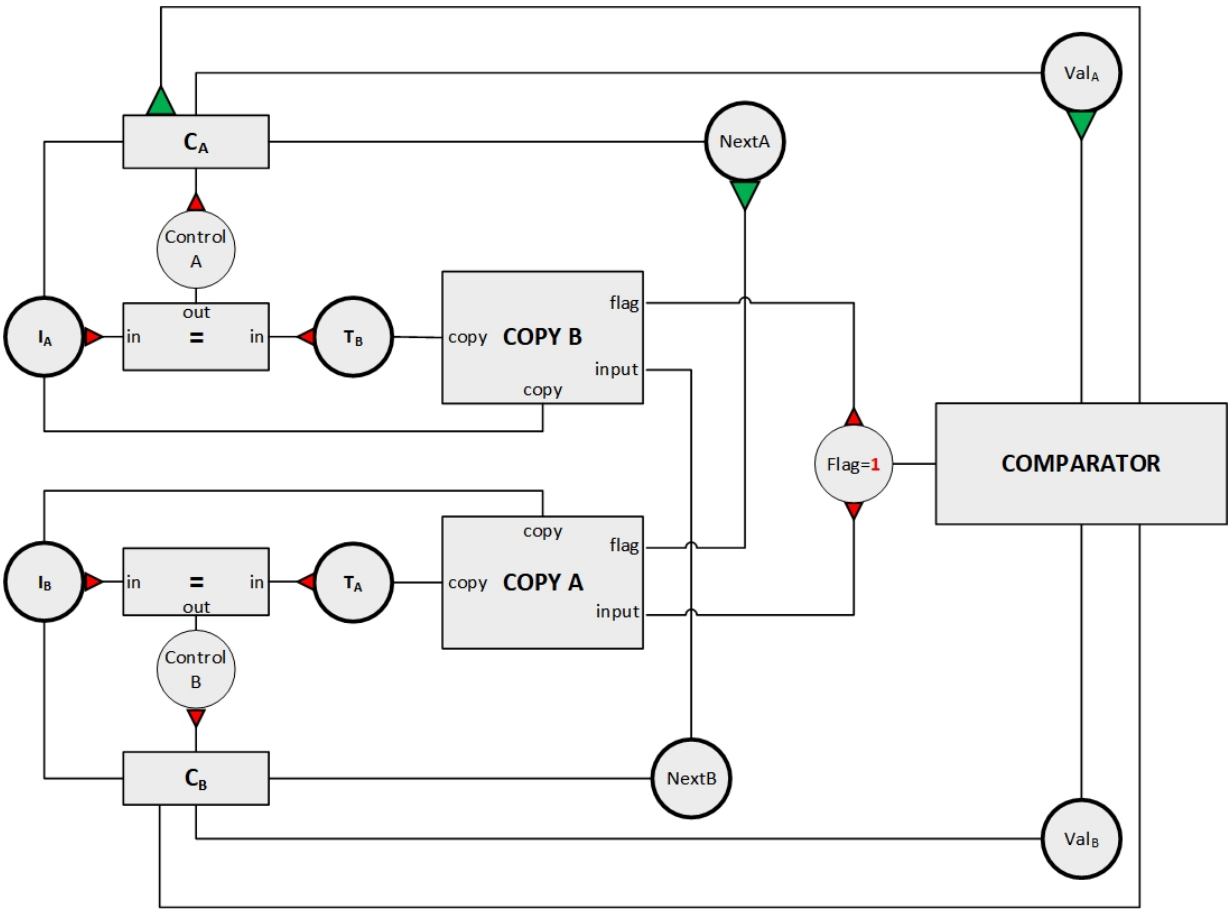

Figure 4.5: Since $Flag = 1$ any internal node of the C_2 gadget has 0 bias with respect to all the other gadgets. As a result, Theorem 4.2.1 applies.

Lemma 4.2.6. *At any equilibrium point of the* LocalNodeMaxCut *instance of Figure 4.3:*

- *If* Flag $= 1$ *then any node in* NextA *experience* 0 *bias with respect to the* CopyA *gadget.*
- *If* Flag $= 0$ *then then any node in* NextB *experience* 0 *bias with respect to the* CopyB *gadget.*

Lemma 4.2.7. *Let an equilibrium of the instance of* LocalNodeMaxCut *of Figure 4.3:*

- *If* Flag $= 1$ *then all nodes of C_A experience* 0 *bias to the* Comparator *gadget.*
- *If* Flag $= 0$ *then all nodes of C_B experience* 0 *bias to the* Comparator *gadget.*

The reason that in Lemma 4.2.7 we refer to all nodes of C_A (respectively C_B) and not just to the nodes in Val_A (respectively Val_B) is that in the constructed instance of LocalNodeMaxCut of Figure 4.3, we connect internal nodes of the C_A gadget with internal nodes of the *Comparator* gadget. This is the only point in our construction where internal nodes of different gadgets share an edge and is denoted in Figure 4.3 and 4.5 with the direct edge between the C_A gadget and the *Comparator* gadget. Now using Lemma 4.2.6 and Lemma 4.2.7 we can prove the correctness of the output nodes *NextA*, Val_A when $Flag = 1$ i.e. $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_B)$ (symmetrically for the nodes in *NextB*, Val_B when $Flag = 0$).

Lemma 4.2.8. *Let an equilibrium of the instance of* LocalNodeMaxCut *of Figure 4.3:*

- *If* Flag $= 1$ *then* NextA $=$ Real-Next(I_A), $\text{Val}_A =$ Real-Val(I_A).
- *If* Flag $= 0$ *then* NextB $=$ Real-Next(I_B), $\text{Val}_B =$ Real-Val(I_B).

Proof. We assume that $Flag = 1$ (for $Flag = 0$ the exact same arguments hold). By Theorem 4.2.2 we have $I_A = NextB$ and by Lemma 4.2.4 we have that $T_B = NextB$. As a result, $I_A = T_B$ and by Lemma 4.2.3 $ControlA = 1$. Lemma 4.2.6 and Lemma 4.2.7 guarantee that the nodes in $NextA$, Val_A of C_A experience 0 bias towards all the other gadgets of the construction and since $ControlA = 1$, we can apply Case 1 of Theorem 4.2.1 i.e. $Val_A = Real\text{-}Val(I_A)$ and $NextA = Real\text{-}Next(I_A)$. ■

Up next we deal with the correctness of the values of the output nodes in Val_B and $NextB$ when $Flag = 1$. We remind again that, even if at an equilibrium $ControlB = 1$, we could not be sure about the correctness of the values of these output nodes due to the bias their neighbors in the *CopyB* and the *Comparator* gadget (Theorem 4.2.1 does not apply). The *Comparator* gadget plays a crucial role in solving this last problem. Namely, it also checks whether the output nodes in *NextB* have correct values with respect to the input I_B and if it detects incorrectness it outputs 0. This is done by the connection of some specific internal nodes of the C_A, C_B gadgets with the internal nodes of the *Comparator* gadget (Figure 4.3: edges between C_A, C_B and *Comparator*).

Lemma 4.2.9. *At any equilibrium of the* NodeMaxCut *instance of Figure 4.3:*

- *If* Flag $= 1$ *then* NextB $=$ Real-Next(I_B)
- *If* Flag $= 0$ *then* NextA $=$ Real-Next(I_A)

We highlight that the correctness of values of the output nodes *NextB*, i.e. $NextB = Real\text{-}Next(I_B)$, is not guaranteed by application of Theorem 4.2.1 (as in the case of correctness of $NextA, Val_A$), but from the construction of the *Comparator* gadget. Lemma 4.2.9 is proven in Section 4.3.3 where the exact construction of this gadget is presented. Notice that Lemma 4.2.9 says nothing about the correctness of the values of the output nodes in Val_B. As we latter explain this cannot be guaranteed in our construction. Surprisingly enough, the *Comparator* outputs the right outcome of the predicate $(Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B))$ even if $Val_B \neq Real\text{-}Val(I_B)$. The latter is one of our main technical contributions in the reduction that reveals the difficulty of LocalNodeMaxCut. The crucial differences between our *Comparator* and the *Comparator* of the previous reductions are discussed in the end of the section. Lemma 4.2.10 formally states the robustness of the outcome of the *Comparator* even with "wrong values" in the nodes of Val_B and is proven in the next section.

Lemma 4.2.10. *At any equilibrium of the* NodeMaxCut *instance of Figure 4.3:*

- *If* $Flag = 1$, NextA $=$ Real-Next(I_A), Val$_A$ $=$ Real-Val(I_A) *and* NextB $=$ Real-Next(I_B) *then*

$$Real - Val(I_A) \leq Real - Val(I_B)$$

- *If* $Flag = 0$, NextB $=$ Real-Next(I_B), Val$_B$ $=$ Real-Val(I_B) *and* NextA $=$ Real-Next(I_A) *then*

$$Real - Val(I_B) \leq Real - Val(I_A)$$

We are now ready to prove Theorem 4.2.5.

Proof of Theorem 4.2.5. Let an equilibrium of the instance of Figure 4.3 with $Flag = 1$ (respectively for $Flag = 0$). By Lemma 4.2.9, $NextB = Real\text{-}Next(I_B)$ and thus Case 1 is established. Moreover by Lemma 4.2.8, $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_A)$. As a result, Lemma 4.2.10 applies and $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$ (Case 2 of Theorem 4.2.5) ■

Having established Theorem 4.2.2 and 4.2.5 the *PLS*-completeness of NodeMaxCut follows easily. For the sake of completeness we present the proof of Theorem 4.2.11 that we describe in the beginning of the section.

Theorem 4.2.11. NodeMaxCut *is PLS-complete.*

Proof. For a given circuit C of the CircuitFlip, we can construct in polynomial time the instance of LocalNodeMaxCut of Figure 4.3. Let an equilibrium of this instance. Without loss of generality, we assume that $Flag = 1$. Then, by Theorem 4.2.2 and Theorem 4.2.5, $I_A = NextB$, $NextB = Real - Next(I_B)$ and $Real - Val(I_A) \leq Real - Val(I_B)$. As a result, we have that

$$\begin{aligned} Real - Val(I_B) &\geq Real - Val(I_A) \\ &= Real - Val(NextB) \\ &= Real - Val\left(Real - Next(I_B)\right) \end{aligned}$$

But if $I_B \neq Real - Next(I_B)$, by Definition 4.2.1 then $Real - Val(I_B) > Real - Val\left(Real - Next(I_B)\right)$ which is a contradiction. Thus $I_B = Real - Next(I_B) = I_B$, meaning that the string defined by the values of I_B is a locally optimal solution for the CircuitFlip problem. ■

4.3 Proofs of the individual gadgets

In the following sections we fully present all the details of the construction for the proof of Theorem 4.2.11. Recall that our NodeMaxCut instance is composed of the following gadgets:

1. *Leverage* gadgets that are used to transmit nonzero bias to nodes of high weight.
2. Two *Circuit Computing* gadgets A,B that calculate the values and next neighbors of solutions.
3. A *Comparator* gadget
4. Two *Copy* gadgets that transfer the solution of one circuit to ther other, and vice versa.
5. Two controller gadgets that decide which circuit should enter *write* or *compute* mode.

Note that whenever we wish to have a node of higher weight that dominates all other nodes of lower weight, we multiply its weight with 2^{kN} for some constant k. We then choose N sufficiently high so that, for all k, nodes of weight 2^{kN} dominate all nodes of weight $2^{(k-1)N}$. Henceforth, we will assume N has been chosen sufficiently high for this purpose.

Moreover, when we have constant nodes of a certain value (i.e. pinned to 1) we connect them with one of two *supernodes*. Supernodes are nodes of huge weight that share an edge and as a result at any equilibrium these nodes have opposite value. In particular, these supernodes have weight 2^{1000N} which dominates the weight of any other node, given we chose N as described above. The term $Control = 1$ means that the $Control$ node has the same value with the 1-node.

When we reference the value of a circuit, we will mean the value that the underlying CircuitFlip instance would output given the same input.

When we reference the value of a node we will mean the side of the cut it lies on. There are two values, $0, 1$ for each side of the cut.

4.3.1 Circuit Computing Gadget

Each of the two computing circuits is meant to both calculate the value of the underlying CircuitFlip instance, as well as the best neighboring solution. For technical reasons one of the two circuits will need to output the complement of the value instead of the value itself, so that comparison can be achieved later with a single node.

In this section we present the gadgets that implement the above circuits in a NodeMaxCut instance. The construction below is similar to the constructions of Schäffer and Yannakakis used to prove MaxCut PLS-complete ([Sch91]). Since NOR is functionally complete we can implement any circuit with a combination of NOR gates. In particular, each NOR gate is composed of the gadgets below. Each such gadget is parameterized by a variable n, and a NOR gadget with parameter n is denoted NOR(n). Since we wish for earlier gates to dominate later gates we order the gates in reverse topological order, so as to never have a higher numbered gate depend on a lower numbered gate. The ith gate in this ordering corresponds to a gadget $NOR(2^{N+i})$. Note that the first gates of the circuit have high indices, while the final gates have the least indices.

We take care to number the gates so that the gates that each output the final bit of the value of the circuit are numbered with the n lowest indexes, i.e. the gate of the kth bit of the value corresponds to

a gate $NOR(2^{N+k})$. This is necessary so that their output nodes can be used for comparing the binary values of the outputs.

The input nodes of these gadgets are either an input node to the whole circuit or they are the output node of another NOR gate, in which case they have the weight prescribed by the previous NOR gate. The input nodes of the entire circuit (which are not the output nodes of any NOR gate) are given weight 2^{5N}.

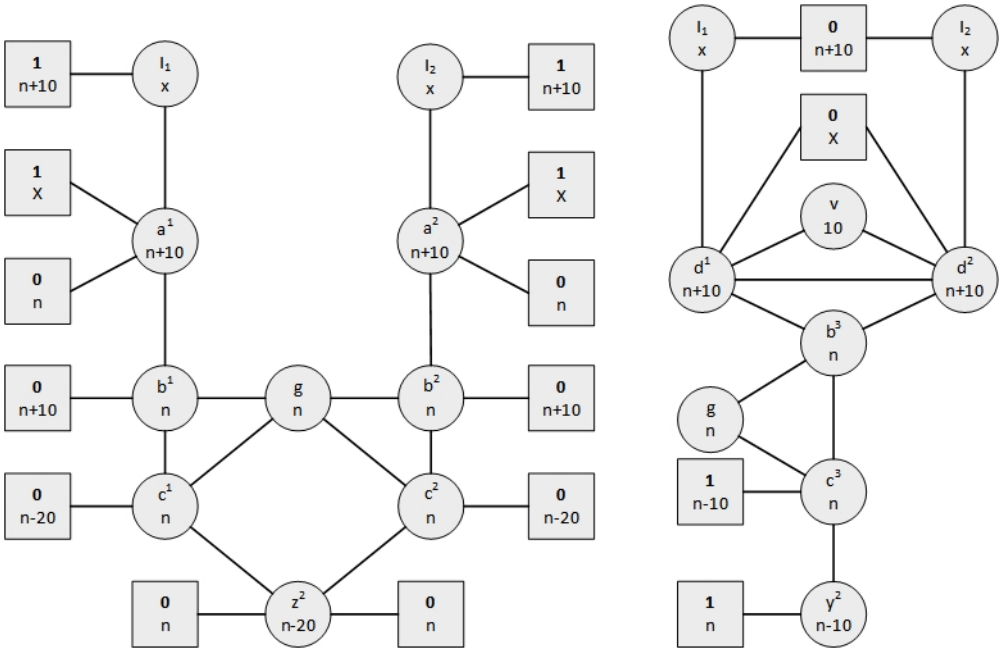

Figure 4.6: The NodeMaxCut *instance implementing a NOR(n) gadget.*

Moreover, we have y_i^1,z_i^1 nodes which are meant to bias the internal nodes of each gadget and determine its functionality. Specifically, $a_i^1, a_i^2, c_i^1, c_i^2, v_i, b_i^3$ are biased to have the same value as y_i^1, while $b_i^1, b_i^2, d_i^1, d_i^2, c_i^3$ are biased to have the same value as z_i^1. This is achieved by auxiliary nodes of weight 2^{-200N}, shown in Figure 4.7.

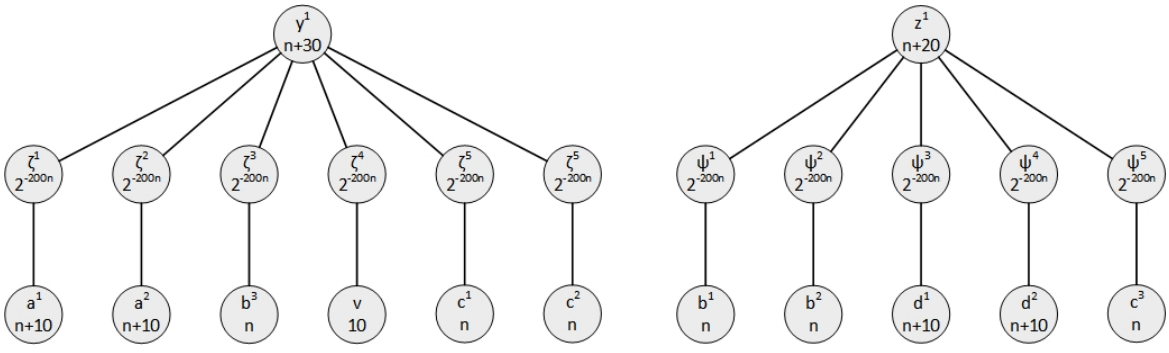

Figure 4.7: Local bias to internal nodes from y_i^1, z_i^1

We also have auxiliary nodes ρ of weight 2^{-500N} that bias the output node g_i to the correct NOR output value. Note that these nodes have the lowest weight in the entire construction.

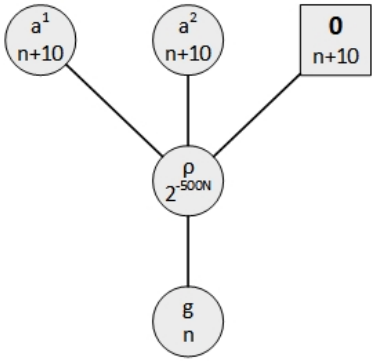

Figure 4.8: Extremely small bias to NOR output value.

These control nodes, y^1, z^1, y^2, z^2 are meant to decide the functionality of the gadget. We say that the y, z nodes have their natural value when $y = 1$ and $z = 0$. We say they have their unnatural value when $y = 0$ or $z = 0$. In general, when these nodes all have their natural values the NOR gadget is calculating correctly and when they have their unnatural values the circuit's inputs are indifferent to the gadget.

Unlike Schäffer and Yannakakis ([Sch91]) we add two extra control variable nodes y^3, z^3 to each such NOR gadget, both of weight $n - 50$. The reason is to ascertain that in case of incorrect calculation at least one y variable will have its unnatural value. Otherwise, it would be possible, for example, to have an incorrect calculation with only z^2 being in an unnatural state.

These NOR gadgets are not used in isolation, but instead compose a larger computing circuit. As Schäffer and Yannakakis do ([Sch91]), we connect each of the control variables z^i, y^i of the above construction so as to propagate their natural or unnatural values depending on the situation. The connection of these gadgets is done according to the ordering we established earlier. Recall that the last m gates correspond to gadgets calculating the value bits, the n gates before them correspond to the output gates of the next neighbor, and the rest are internal gates of the circuit.

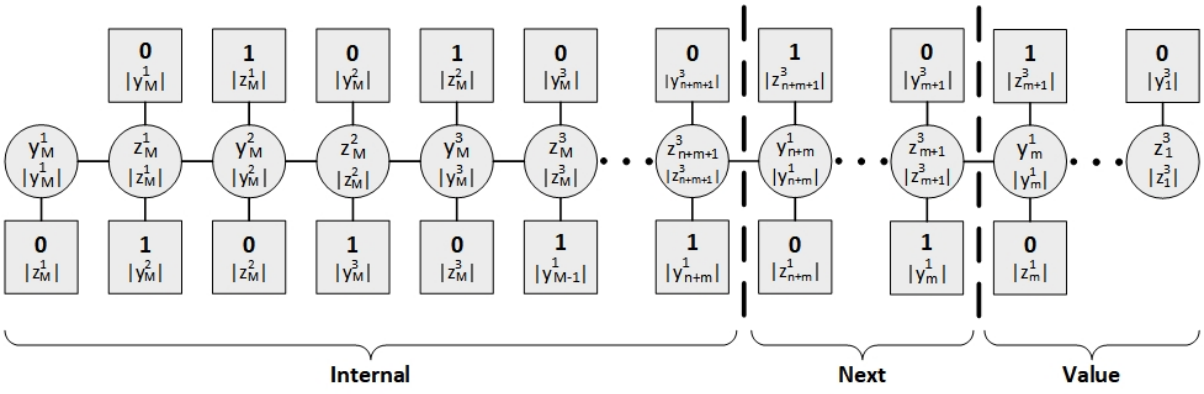

Figure 4.9: Connecting the control nodes of the NOR gadgets. Recall that M is the number of total gates in the circuit, n is the number of solution bits and m is the number of value bits. Note that the gates are ordered in reverse, i.e the first gates have highest index.

These gadgets' function is twofold. Firstly, they detect a potential error in a NOR calculation and propagate it to further gates, if the control variables have their unnatural values. Second,if the control variables have their unnatural values, they insulate the inputs so that they are indifferent to the gadget and can be changed by any external slight bias.

Furthermore, all the nodes of these gadgets are all multiplied by a 2^{100N} weight, except the nodes of the NOR gadget corresponding to the final bits of the value which are multiplied by 2^{90N}. This is so that a possible error in the calculation of the next best neighbor supersedes any possible result of the comparison. The auxiliary nodes introduced above, which are meant to induce small biases to internal

nodes, are not multiplied by anything.

Lastly, for technical simplicity, we have a single node for each computing circuit meant to induce bias to all control variable nodes y, z at the same time. The topology of the connection is presented below.

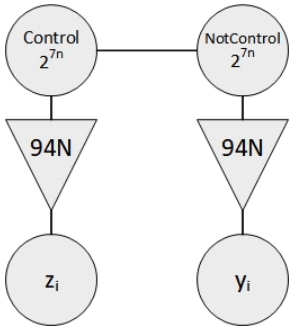

Figure 4.10: We use a single node Control to bias all control nodes y, z. Note that this node is connected with the y, z nodes through leverage gadgets

We now state the properties of these gadgets.

Lemma 4.3.1. *In an equilibrium, if $z_i^1 = 1$ and $y_i^1 = 0$, then $I_1(g_i)$, $I_2(g_i)$ are indifferent with respect to the gadget G_i.*

Lemma 4.3.2. *If gate G_i is incorrect, then $z_i^2 = 1$. If $y_i^2 = 0$ then $z_i^2 = 1$. If $z_i^2 = 1$, then for all $j < i$ $z_j^1 = z_j^2 = z_j^3 = 1$ and $y_j^1 = y_j^2 = y_j^3 = 0$.*

Lemma 4.3.3. *Suppose $z_i^1 = 0$ and $y_i^1 = 1$. If g_i is correct then z^2 and y^2 are indifferent with respect to the other nodes of the gate G_i. If g_i is incorrect then g_i is indifferent with respect to the other nodes of the gate G_i, but gains the node ρ of weight 2^{-500N}.*

Lemma 4.3.4. *Assuming all nodes of the computing circuit gadget are in equilibrium and have no external biases. If Control $= 1$ then $\forall i, z_i^1 = 0, y_i^1 = 1, z_i^2 = 0, y_i^2 = 1, z_i^3 = 0, y_i^3 = 1$. If Control $= 0$ then $\forall i z_i^1 = 1, y_i^1 = 0, z_i^2 = 1, y_i^2 = 0, z_i^3 = 1, y_i^3 = 0$.*

Lemma 4.3.5. *If Control $= 1$ then all y, z nodes have a 2^{-87N} bias towards their natural values. If Control $= 0$ then all y, z nodes have a 2^{-87N} bias towards their unnatural values.*

Proof. The *NotControl* nodes are dominated by *Control*'s bias of 2^{7N} and hence have the opposite value. By lemma 4.3.15 we have that *Control* and *NotControl* experience at most 2^{6N} bias, while the y, z nodes experience 2^{-87N} bias towards the values opposite *Control* and *NotControl*, which proves the claim. ■

The proofs of these statements can be found in [Sch91] since this particular gadget is unchanged.

Lemma 4.3.6. *Assuming all nodes of the computing circuit gadget are in equilibrium and have no external biases. If Control $= 1$ then $\forall i, z_i^1 = 0, y_i^1 = 1, z_i^2 = 0, y_i^2 = 1, z_i^3 = 0, y_i^3 = 1$. If Control $= 0$ then $\forall i z_i^1 = 1, y_i^1 = 0, z_i^2 = 1, y_i^2 = 0, z_i^3 = 1, y_i^3 = 0$.*

Having stated the above auxiliary lemmas, we can finally prove the theorem specifying the behaviour of our computing circuits.

Theorem 4.3.7. *At any equilibrium of the* LocalNodeMaxCut *of Figure 4.3.*

1. *If Control$\ell = 1$ and the nodes of Nextℓ,Val$_\ell$ experience 0 bias from any other gadget beyond C_ℓ then:*

   - Nextℓ = Real-Next(I_ℓ)
   - Val$_\ell$ = Real-Val(I_ℓ)

2. *If* $Control\ell = 0$ *then each node in* I_ℓ *experiences* 0 *bias from the internal nodes of* C_ℓ.

3. $Control\ell$ *experiences* $w_{Control\ell}$ *bias from the internal nodes of* C_ℓ.

Proof.

1. Since $Control\ell = 1$ and since we assumed no node experiences any external bias, by lemma 4.3.6 we have that all y, z have their natural values and hence all gates compute correctly, by lemma 3.3.2. Therefore, $Next\ell = Real - Next(I_\ell)$ and $Val_\ell = Real - Val(I_\ell)$.

2. Since $Control\ell = 0$, by lemma 4.3.6 all y, z have their unnatural values. Since all NOR gadgets have unnatural control nodes we have that their inputs are indifferent with respect to the gadgets. Hence, the claim that they are unbiased follows.

3. The $Control\ell$ node is connected to a node $NotControl\ell$, of weight $W_{NotControl\ell} = 2^{7N}$, as well as to several leverage gadgets, which contribute bias at most $2^{100N-94N} = 2^{6N}$. Hence, the 2^{7N} bias dominates.

■

4.3.2 Copy Gadget

The *Copy* Gadgets transfer the values of the next best Neighbor of a circuit to the input of the other circuit. This is fundamental for the correct computation of the local optimum. There are some technical conditions that these gadgets should satisfy, which we discuss in the following.

The purpose of the *Copy* Gadgets is twofold. Firstly, when the *Flag* node has value 1, they are meant to give the inputs of Circuit B a slight bias to take the values of the best flip neighbor that Circuit A offers, that is *NextA*. Secondly, in this case they are meant to give zero bias to the output nodes of Circuit A that calculate the best flip neighbors. This is because when node *Flag* is 1, the input of circuit A is going to change, which means that the NOR gates of this circuit will compute the new values. A consequence of the functionality of the NOR gadgets is that the outputs of a gadget are only biased towards the correct value with a very small weight. This is because the gadget is constructed in a way that allows these nodes to be indifferent to all of their neighbors when the time comes to change their value. As a result, if we connect the output nodes with other gadgets, we have to ensure that they will experience zero bias from them in order for the computation to take place properly. Since the outputs of Circuit A that produce the next best neighbor are connected to the *Copy* Gadgets, we should ensure that they will experience zero bias when node *Flag* is 1, so that they can change properly. A similar functionality should be implemented when node *Flag* is 0.

In this Section we present the gadgets that implement the above functionality. There are two *Copy* gadgets with similar topology, *CopyA* and *CopyB*. For simplicity, we only describe the details of *CopyA*. The gadget takes as input the value of node *Flag*, which determines whether a value should be copied or whether the outputs of Circuit A should experience zero bias. It also takes as input *NextA*, which is the next best neighbor calculated by Circuit A. The output of the gadget is a bias to nodes I_B and T_A towards adopting the value of *NextA*.

At this point, one might wonder why we didn't just connect the output of the *CopyA* gadget to the input I_B. This is because the value of I_B also depends on the control variables. If the control variables of the input gates have natural values, then the inputs experience great bias from the gate, making it impossible for their values to change by the *Copy* Gadget. Hence, the *Copy* Gadget gives a slight bias to node T_A, which is an input to an auxiliary circuit that compares it with I_B (i.e the *Equality* gadget) . If they are not equal, this means that the output has not been transferred yet. In this case, the output of the gadget is given a suitable value to bias the control nodes towards unnatural values. When this happens, the inputs I_B can change to the appropriate values.

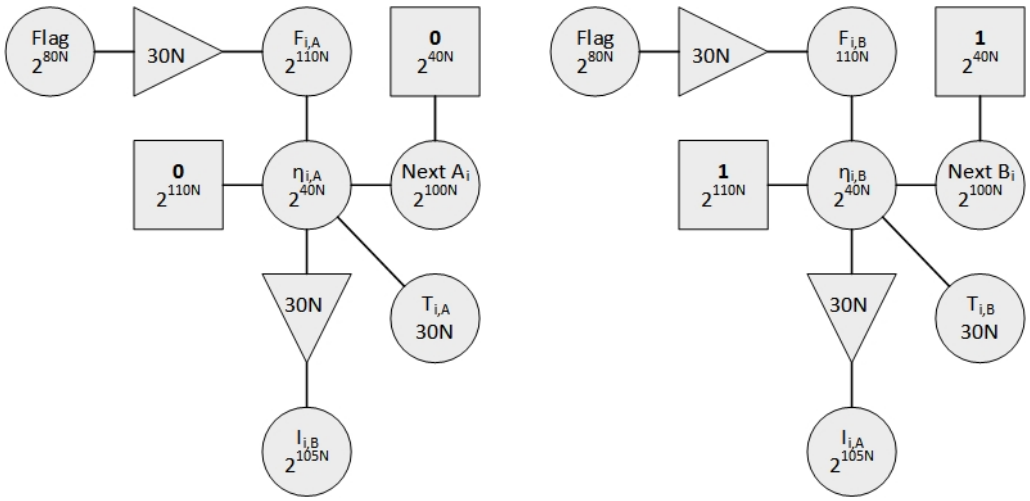

Figure 4.11: The gadgets that copy the values from one circuit to the other

Note that we have one of the above gadgets for each of the bits of the next best neighbor solution that the *Circuit Computing* gadgets output.

We have a gadget of Figure 4.11 for each of the m bits of the next best neighbor. Nodes $F_{i,A}$ has a very large weight in order to dominate the behavior of $\eta_{i,A}$. However, we do not want this node to influence the behavior of *Flag*. For this reason, we connect *Flag* with $F_{i,A}$ using a *Leveraging* gadget. Notice that the behavior of $F_{i,A}$ is dominated by *Flag* by weight at least 2^{50N}. Another important point is that we connect the output of the *CopyA* gadget with the input of Circuit B using another *Leveraging* gadget. This is due to the fact that the weight of the input nodes is of the order of 2^{105N}, which is far more than the weight of $\eta_{i,A}$. Hence, we do not want the input nodes to influence the value of $\eta_{i,A}$, while also ensuring that the *Copy* gadget gives a slight bias to the inputs I_B towards the value of *NextA*.

We now prove Lemma 4.2.4, which makes precise the already stated claims about the function of the *Copy* Gadgets.

Lemma 4.3.8. *At any equilibrium point of the* LocalNodeMaxCut *instance of Figure 4.3:*

- *If* Flag $= 1$, *i.e.* NextB writes *on* I_A, *then*
  1. $T_B =$ NextB
  2. *If* $ControlA = 0$ *then* $I_A = T_B =$ NextB
- *If* Flag $= 0$ *i.e.* NextA writes *on* I_B, *then*
  1. $T_A =$ NextA
  2. *If* $ControlB = 0$ *then* $I_B = T_A =$ NextA

Proof. We prove the claim for $Flag = 1$. The case $Flag = 0$ is identical.

We begin with the first claim. Due to the leveraging gadget, node $F_{i,B}$ experiences bias from *Flag* which is slightly less than 2^{50N}. Hence, it is biased towards 0 with weight at least 2^{49N}. This is greater than the weight of $\eta_{i,B}$, which is the other neighbor of $F_{i,B}$. Hence, $F_{i,B} = 0$ at equilibrium. Now node $\eta_{i,B}$ experiences zero total bias from nodes $F_{i,B}$ and constant 1 and biases 2^{100N} by $NextB_i$, 2^{30N} by $T_{i,B}$ and slightly more that 2^{75N} by the input $I_{i,A}$ due to leveraging, which means that its value at equilibrium will be determined by $NextB_i$. Specifically, $\eta_{i,B} = \neg NextB_i$ at equilibrium. Now, node $T_{i,B}$ experiences bias 2^{40N} from $\eta_{i,B}$ and biases of the order of 2^{7N} from the gates of the controller gadget. Hence, $T_{i,B}$ has bias towards $NextB_i$ equal to $w_{\eta_{i,B}}$ and will take this value at equilibrium.

To prove the second claim, we use the already proven fact that $\eta_{i,B} = \neg NextB_i$ when $Flag = 1$. Due to the *Leverage* gadget, node I_i experiences bias slightly less than 2^{10N} from node $\eta_{i,B}$. Since $ControlA = 0$, by Lemma 4.2.1, we have that $I_{i,A}$ is indifferent with respect to the gadget C_A, and will therefore take the value of $\neg\eta_{i,B} = NextB_i = T_{i,B}$

■

Lemma 4.3.9. *At any equilibrium point of the* NodeMaxCut *instance of Figure 4.3:*

1. *If* Flag $= 1$ *then any node in* NextA *experience* 0 *bias with respect to the* CopyA *gadget.*
2. *If* Flag $= 0$ *then then any node in* NextB *experience* 0 *bias with respect to the* CopyB *gadget.*

Proof. We notice that due to leveraging, node $F_{i,A}$ of gadget *CopyA* experiences bias slightly less than 2^{50N} from node *Flag* $= 1$. This dominates its behavior, since the other neighbor $\eta_{i,A}$ has weight that is orders of magnitude smaller. Hence, $F_{i,A} = 0$. Now, node η_i, experiences total bias $2 * 2^{110N}$ from nodes $F_{i,A}$ and constant 0, 2^{100N} from $NextA_i$, 2^{30N} from $T_{i,A}$ and slightly more than 2^{75N} from $I_{i,B}$ due to the *Leverage* gadget used. This means that $\eta_{i,A} = 1$. Now we are ready to prove our claim. Node $NextA_i$ is connected to nodes $\eta_{i,A}$ and constant 0 of gadget $CopyA_i$. They have the same weight and opposite values at equilibrium. This means that $NextA_i$ has 0 bias with respect to $CopyA_i$, i.e it is indifferent.

The case for *Flag* $= 0$ follows symmetrically. ■

4.3.3 Comparator gadget

The purpose of the *Comparator* gadget is to implement the binary comparison between the bits of the values of the two circuits. At the same time we need to ensure that the nodes of the losing circuit (i.e the circuit with the lower value) are indifferent with respect to the *Comparator* gadget, so that Lemma 4.2.1 can be applied.

In particular, the output nodes that correspond to the bits of the value, presented in the section if the computing circuits, with weights $2^{90N} * 2^{N+i}$ are each connected as below.

Note that the output bits of the second circuit B are the complement of their true values, in order to achieve comparison with a single bit. The weight of the *Flag* node is 2^{80N}

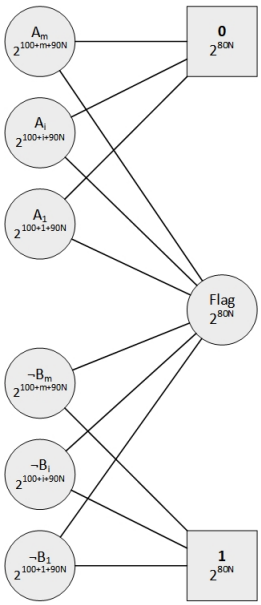

Figure 4.12: Nodes of the Comparator *gadget. Note that Circuit B is meant to output the complement of its true output.*

To see why the value of node *Flag* implements binary comparison one needs to consider four cases: In the first two, where the ith bits are both equal, the total bias *Flag* experiences is zero, since it experiences bias towards a certain bit as well as the complement of said bit. In the other two, where one bit is 1 and the other is 0, the *Flag* node will experience 2^i bias towards either value, which will supersede all lower bits.

However, the *Comparator* gadget is meant not only to implement comparison between values, but also to detect whether a circuit is computing wrongly and, hence, to fix it. To this end we connect the following control nodes to the node *Flag*: the control nodes $y^3_{m+1,A}$ for circuit A and $z^3_{m+1,B}$ for circuit B, where $m + 1$ is the last NOR gadget before the bits of the values (recall that we have m

value bits and that $w_{y^3_{i,A}} = w_{z^3_{i,B}} = 2^{100N} * (2^{N+m+1} - 50))$ (see Figure 4.9), as well as the control nodes $y^3_{i,A}, z^3_{i,B}, \forall i \leq m$ for each NOR gadget that corresponds to an output bit of the value (which have weight $w_{y^3_{i,A}} = w_{z^3_{i,B}} = 2^{90N} * (2^{N+i} - 50))$. The nodes $y^3_{m+1,A}$ and $z^3_{m+1,B}$ are used to check whether the next best neighbor has been correctly computed. If it isn't, these nodes dominate *Flag*, due to their large weight of 2^{100N} compared to the weight of the value bits, which is of the order of 2^{90N}. The control nodes of the output bits of the value are used in a more intricate way to ensure that even if one to the results is not correct, the output of the comparison is the desired one. Details are provided in Lemma 4.3.11. All these nodes are connected in such a way that a control node with unnatural value, biases *Flag* towards fixing that circuit.

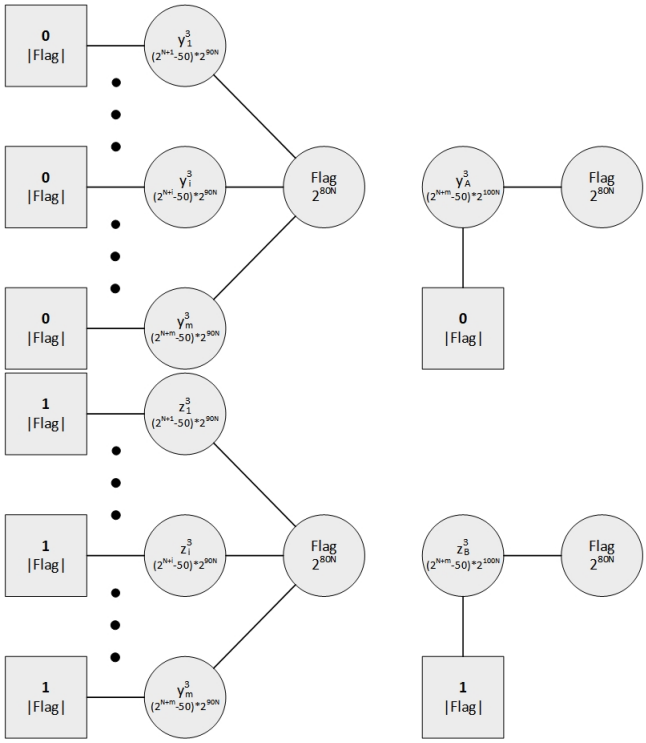

Figure 4.13: Connection between the control nodes *and the* Flag *node.*

We prove the following properties:

Lemma 4.3.10. *Let an equilibrium of the instance of* NodeMaxCut *of Figure 4.3:*

- *If* Flag $=$ *1 then all nodes of* C_A *experience* 0 *bias to the* Comparator *gadget.*
- *If* Flag $=$ *0 then all nodes of* C_B *experience* 0 *bias to the* Comparator *gadget.*

Proof. Suppose *Flag* $= 1$. Then the only nodes of C_A connected to the *Comparator* gadget are the value output bits and certain control nodes, in such a way that they are connected to either *Flag* or a constant node 0 of weight equal to *Flag*. In all cases, both biases cancel each other out and the nodes of Circuit A are indifferent. Suppose *Flag* $= 0$. Then the only nodes of C_B connected to *Flag* are also connected with a constant 1 node. Similarly to the first case, all nodes of circuit B are indifferent with respect to the *Comparator* gadget when *Flag* $= 0$. ■

We now prove the most important lemma of the *Comparator* gadget. Our goal is to compare the output values of the two circuits, so that we change the input of the circuit with the smaller real value. The main difficulty lies in that one or both of the circuits might produce incorrect bits in their output. A simple idea would be to try to detect any incorrect output bits and influence *Flag* accordingly, as we do with control variables $y^3_{m+1,A}$ and $z^3_{m+1,B}$. However, if the least significant bit of a circuit is incorrect, the weight of the corresponding control node is exponentially smaller that the rest of the bits. Hence, it

cannot dominate the outcome of the comparison. This means that sometimes we might be in equilibrium where some output nodes are incorrect. To alleviate this problem we propose this construction.

The idea behind this lemma is very simple: if it is guaranteed that the output of one of the circuits is correct and we know which bits of the other circuit *might* be wrong, we can still compare their true values. This is accomplished by an extension of the traditional comparison method, by also taking into account the control variables of the output bits and examining all the possible cases. This lemma is very useful in our proof, since by Lemma 4.2.8 we know that at least one of the circuits computes correctly in equilibrium.

Lemma 4.3.11. *At any equilibrium:*

Suppose Flag $= 1$. *If* $\forall i, z^1_{i,A} = 0, y^1_{i,A} = 1, z^2_{i,A} = 0, y^2_{i,A} = 1, z^3_{i,A} = 0, y^3_{i,A} = 1$ *and* $\forall i > m, z^1_{i,B} = 0, y^1_{i,B} = 1, z^2_{i,B} = 0, y^2_{i,B} = 1, z^3_{i,B} = 0, y^3_{i,B} = 1$ *then* $Real - value(I_A) \leq Real - value(I_B)$

Suppose Flag $= 0$. *If* $\forall i, z^1_{i,B} = 0, y^1_{i,B} = 1, z^2_{i,B} = 0, y^2_{i,B} = 1, z^3_{i,B} = 0, y^3_{i,B} = 1$ *and* $\forall i > m, z^1_{i,A} = 0, y^1_{i,A} = 1, z^2_{i,A} = 0, y^2_{i,A} = 1, z^3_{i,A} = 0, y^3_{i,A} = 1$ *then* $Real - value(I_B) \leq Real - value(I_A)$

Proof. Since for all gates that do not correspond to value bits (see Figure 4.9), we have that they possess natural values, and hence *Flag* is indifferent with respect to them, we only need to examine the final m gates that correspond to the value bits.

We denote the kth bit of Val_A, Val_B as A_k, B_k. B_k corresponds to the actual value of the kth bit of the circuit B instead of its complement for simplicity. The actual value of the node corresponding to B_k is the opposite. We also denote $z^2_{k,B}$ the control node corresponding to the bit B_k. We make the distinction between A_k, B_k and $Real(A_k), Real(B_k)$. These may be equal or different depending on whether the circuit calculated the kth bit correctly. By the assumption we know that $A_k = Real(A_k)$ since A calculates correctly. We do not know whether $B_k = Real(B_k)$, but we do know that $B_k \neq Real(B_k) \implies z^2_{k,B} = 1$.

We consider three cases.

In the case that $(A_k, B_k, z^2_{k,B}) \in (0,0,0), (1,1,0), (0,1,1)$, *Flag* experiences bias at most $2 * 2^{90N} * (50)$ from this bit towards $Flag = 1$ in any of these cases. In this case, we have that either $Real(A_k) = Real(B_k)$ or $Real(A_k) < Real(B_k)$, depending on whether B_k calculated correctly. Either way, $Real(A_k) \leq Real(B_k)$.

In the case that $(A_k, B_k, z^2_{k,B}) \in (0,1,0)$, *Flag* experiences bias $2 * 2^{90N} * (2^{N+k})$ from this bit towards $Flag = 1$. In this case, we have that $Real(A_k) < Real(B_k)$, since both calculate correctly.

In the case that $(A_k, B_k, z^2_{k,B}) \in (0,0,1), (1,0,0), (1,1,1), (1,0,1)$ then *Flag* experiences bias at least $2 * 2^{90N} * (2^{N+k} - 50)$ towards $Flag = 0$ from this bit in any of these cases. In these cases, $Real(A_k)$ might be higher, but we will show that these cases can never matter.

Suppose k the highest i for which $(A_i, B_i, z^2_{k,B}) \notin (0,0,0), (1,1,0), (0,1,1)$.

If no such k exists then all bits must lie in the first case and hence $\forall k Real(A_k) \leq Real(B_k)$. Hence, $Real - value(I_A) \leq Real - value(I_B)$.

If for that k, $(A_k, B_k, z^2_{k,B}) \in (0,1,0)$, we know that $Real(A_k) < Real(B_k)$ while for all higher bits $k Real(A_k) \leq Real(B_k)$. This means that $Real - value(I_A) < Real - value(I_B)$, since the lower bits don't matter as long as we have a strict inequality in a high bit.

Lastly, if we have that $(A_k, B_k, z^2_{k,B}) \in (0,0,1), (1,0,0), (1,1,1), (1,0,1)$, we have that *Flag* experiences bias at least $2 * 2^{90N} * (2^{N+k} - 50)$ towards $Flag = 0$ from this bit. Furthermore, it experiences bias at most $2 * 2^{90N} * (50)$ towards $Flag = 1$ from each bit higher that k. Each bit i lower than k causes bias at most $2 * 2^{90N} * (2^{N+i})$ each towards $Flag = 1$. In total, if we have m bits, we have at most $(m-k) * 2 * 2^{90N} * (50) + \sum_{i<k} 2 * 2^{90N} * (2^{N+i}) \leq (m) * 2 * 2^{90N} * (50) + 2 * 2^{90N} * (2^{N+k} - 2^N)$ towards $Flag = 1$ and at least $2 * 2^{90N} * (2^{N+k} - 50)$ towards $Flag = 0$. For N sufficiently high, the bias towards 0 would win, making *Flag* no longer have 1 as its best response, which is a contradiction. Hence, the third case can not happen in an equilibrium with $Flag = 1$.

The case for $Flag = 0$ is identical, with the only difference being we consider $y^2_{k,}$ instead. ■

Lemma 4.3.12. *At any equilibrium of the* NodeMaxCut *instance of Figure 4.3:*

- *If* Flag $= 1$ *then* NextB $=$ Real-Next(I_B)
- *If* Flag $= 0$ *then* NextA $=$ Real-Next(I_A)

Proof. Assume an equilibrium with $Flag = 1$ and $NextB \neq Real - Next(I_B)$. By Lemma 3.3.2 we have that Circuit B is computing incorrectly and hence the control node $z^3_{m+1,B}$ (i.e. the last gate before the value bits) has its unnatural value, which is $z^3_{m+1,B} = 1$.

Assume that, $ControlA = 0$. Then by Lemma 4.2.4 we have that $I_A = T_B$, which by Lemma 4.2.3 we have $ControlA = 1$, a contradiction. Hence, $ControlA = 1$.

Therefore, since have that $ControlA = 1$ and that $NextA = Real - Next(I_A)$, which by Lemma 4.3.6, implies that the corresponding node $y^3_{m+1,A}$ has its natural value $y^3_{m+1,A} = 1$.

This means that *Flag* experiences bias towards 0 at least $2 * 2^{100N} * (2^{N+m+1} - 50)$ from the nodes $z^3_{m+1,B}$,$y^3_{m+1,A}$, which dominates *Flag* to take value 0. This is a contradiction since we assumed that $Flag = 1$ at equilibrium. Hence, if $Flag = 1$ then it must be that $NextB = Real - Next(I_B)$.

Similarly, we can prove that if $Flag = 0$ then $NextA = Real - Next(I_A)$ ■

Lemma 4.3.13. *At any equilibrium of the* LocalNodeMaxCut *instance of Figure 4.3:*

- *If* $Flag = 1$, NextA = Real-Next(I_A), Val_A = Real-Val(I_A) *and* NextB = Real-Next(I_B) *then*

$$Real - Val(I_A) \leq Real - Val(I_B)$$

- *If* $Flag = 0$, NextB = Real-Next(I_B), Val_B = Real-Val(I_B) *and* NextA = Real-Next(I_A) *then*

$$Real - Val(I_B) \leq Real - Val(I_A)$$

Proof. Assume that, $ControlA = 0$. Then by Lemma 4.2.4 we have that $I_A = T_B$, which by Lemma 4.2.3 we have $ControlA = 1$, a contradiction. Hence, $ControlA = 1$.

Since $Flag = 1$ and we have that $ControlA = 1$, by Lemma 4.3.6 all control nodes of C_A have their natural values. Furthermore, since by the proof of Lemma 4.2.9 we know that all control nodes of weight 2^100N have their natural values, we can apply Lemma 4.3.11. Therefore, $Real - Val(I_A) \leq Real - Val(I_B)$

The proof for $Flag = 0$ is identical.

■

4.3.4 Equality Gadget

The *Equality* Gadgets are used to check whether the next best neighbor of a circuit has been successfully transferred to the input of the other circuit. The output of the *Equality* gadget is connected to the control variables of the circuit that should receive the new input. If the new input has not been transferred, the output of this gadget biases the *Control* node towards 0, which biases the internal control nodes towards unnatural values. This enables the inputs of the circuit to change successfully to the next solution. When the new solution is transferred, the output of the gadget changes, in order to bias the control nodes towards their natural values, so that the computation can take place.

Since we have two possible directions, both from Circuit A to Circuit B and vice versa we need two copies of the gadgets described in this section.

We will now describe the function of the *Equality* Gadget when Circuit A gives feedback to Circuit B. The *Equality* Gadget takes as inputs the T_A nodes from the *CopyA* Gadgets and I_B and simply checks whether they are equal. Due to Lemma 4.2.4, in equilibrium the T_A nodes have the same value as *NextA* which we want to transfer. One might try to connect *NextA* as input to the *Equality* gadget. The reason we avoid this construction is that we do not want the output nodes of the *Circuit Computing* gadget C_A to experience any bias from this gadget, because the computation changes their value with very small bias. For this reason, we connect T_A nodes to the input that are dominated by η_A nodes. The input nodes I_B are dominated by either the nodes in the NOR gadgets or η_A, hence we can connect them directly as inputs to the gadget.

For each bit of the next best neighbor, we construct a gadget as in Figure 4.14, which performs the equality check for the $i-th$ bit of the next best neighbor. The idea for this construction is very simple: the weights decrease as we come closer to the output, so that the input values dominate the final result. If the inputs are equal, the final value will be 0. Notice that we have put and intermediate node between I_B and the gadget to ensure that the two input nodes will have equal weight. A detailed analysis is provided in the proof of Lemma 4.2.3.

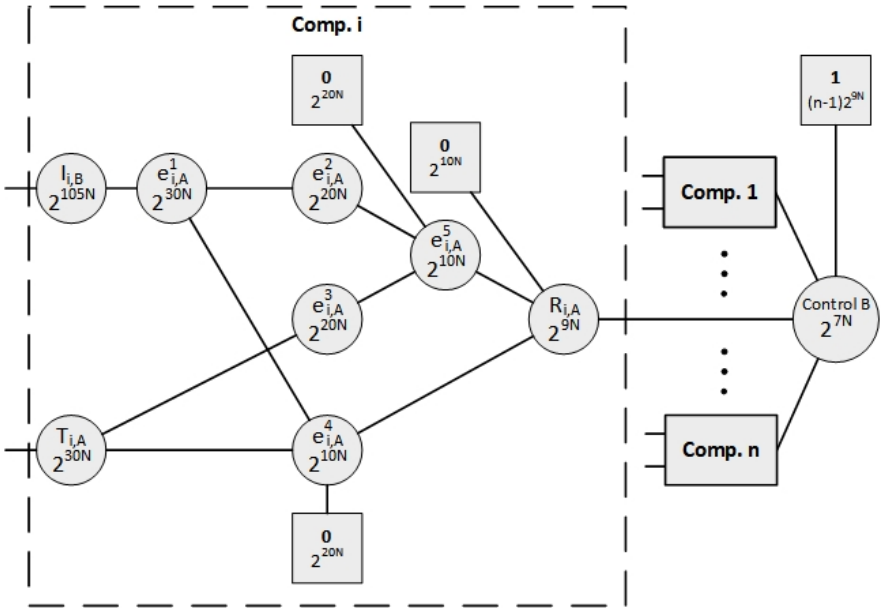

Figure 4.14: This gadget performs equality check for the bits $I_{i,B}$ and $T_{i,A}$. If they are equal, $R_{i,A} = 0$ in equilibrium. We have n such gadgets for each of the two circuits. The n gadgets are connected to produce the final output, which is ControlB.

Now that we have gadgets to perform bit wise equality checks, we need to connect them all to produce the output of the *Equality* gadget. This is done by the construction of Figure 4.14 Essentially, the idea is that if all the bits are equal, all the comparison results will be 0 and will dominate the *ControlB* to take 1. If at least one result is 1, then together with the constant node 1 will bias *ControlB* to take value 0.

We now prove the main lemma concerning the *Equality* Gadget, which states that in equilibrium, the output of the *Equality* will be 1 if and only if the two inputs to the gadget are equal.

Lemma 4.3.14. *Let an equilibrium of the overall* NodeMaxCut *instance of Figure 4.3. Then*

- $ControlA = (I_A = T_B)$
- $ControlB = (I_B = T_A)$

Proof. For simplicity we only prove the second claim, since the first follows by similar arguments. We first focus on on the behavior of a single *Equality* gadget. We would like to prove that $R_{i,A} = 0$ if and only if $I_{i,B} = T_{i,A}$.

We first observe that node $e^1_{i,A}$ is biased with weight 2^{105N} by $I_{i,B}$, which is greater that the bias from its other neighbor $e^2_{i,A}$. Hence, in equilibrium it is always the case that $e^1_{i,A} = \neg I_{i,B}$. Moreover, nodes $e^2_{i,A}$ and $e^3_{i,A}$ essentially function as the complements of $e^1_{i,A}$ and $T_{i,A}$. This is because they are biased with weight 2^{30N} by them, which is greater than the bias by node $e^5_{i,A}$. Hence, $e^2_{i,A} = \neg e^1_{i,A}$ and $e^3_{i,A} = \neg T_{i,A}$.

We first examine the case where $I_B = T_A$. Then $e^1_{i,A} = \neg T_{i,A}$. Since these nodes have equal weights, node $e^4_{i,A}$ experiences 0 total bias from them and is biased by constant node 0 with weight 2^{20N} and by R_2 with weight 2^{9N}. Therefore, $e^4_{i,A} = 1$. By the previous observations we have that $e^2_{i,A}$ and $e^3_{i,A}$ have opposite values, which means that $e^5_{i,A}$ has bias 0 from these two nodes. Is also has bias 2^{20N} by constant 0 and 2^{9N} from $R_{i,A}$. Hence, $e^5_{i,A} = 1$. Nodes $e^4_{i,A}$ and $e_{5.i}$ bias node $R_{i,A}$ towards 0 with weight $2 * 2^{20N}$, which is greater than the bias from constant 0 and *ControlB*. As a result, we have that $R_{i,A} = 0$ and the argument is complete in this case.

Now we examine the case where $I_{i,B} \neq T_{i,A}$. Assume that $I_{i,B} = 1$, the other case follows similarly. Then, $e^1_{i,A} = 0$, $T_{i,A} = 0$, $e^2_{i,A} = 1$, $e^3_{i,A} = 1$. This means that $e^5_{i,A}$ is biased with weight at least $2 * 2^{30N}$ towards 1, which is greater than the combined weight of $R_{i,A}$ and constant 0. Therefore, $e^5_{i,A} = 0$. Now we observe that $R_{i,A}$ is biased with weight at least $2 * 2^{20N}$ towards 1 by nodes $e^5_{i,A}$ and constant 0,

which is greater that the combined weight of $e^4_{i,A}$ and *ControlB*. Hence, $R_{i,A} = 1$ in this case. If $I_{i,B} = 0$, then we could prove similarly that $e^4_{i,A} = 0$, which implies that $R_{i,A} = 1$ by the same argument.

We will now prove that *ControlB* takes the appropriate value. First of all, we observe that *ControlB* is connected with *NotControlB*, (part of the *Circuit Computing* gadget) which has weight 2^{7N} and with $R_{i,A}$ nodes which have weight 2^{9N}. It is also biased with weight slightly more than 2^{6N} by each of the control variables y_i due to the leverage gadget. This means that for N large enough *ControlB* is dominated by the behavior of the $R_{i,A}$ nodes. Suppose that $I_{i,B} = T_{i,A}$ for all i, $1 \leq i \leq n$. By the preceding calculations, we have that $R_{i,A} = 0$ for all i. Hence, $ControlB$ experiences total bias $n * 2^{9N}$ towards 1, which is greater than the weight of constant node 1. Thus, $ControlB = 1$ in this case. Now suppose that there exists a j, $1 \leq j \leq n$, such that $I_{2,j} = \neg T_{j,A}$. By the preceding calculations, $R_{j,A} = 1$. Hence, node *ControlB* is biased by nodes $R_{j,A}$ and constant 1 towards 0 with weight at least $(n-1)*2^{9N}+2^{9N} = n*2^{9N}$, which is greater than the combined weight of all the other $R_{i,A}$'s. Therefore, $ControlB = 0$ in this case and the proof is complete.

■

4.3.5 Leverage Gadget

The *Leverage* gadget is a basic construction in the PLS completeness proof. This gadget solves a basic problem in the reduction. Suppose that we have a node with relatively small weight A and we want to bias a node with large weight B. For example, the large node might be indifferent towards its other neighbors, which would allow even a small bias from the small node to change its state. We would also like to ensure that the large node does not bias the smaller one with very large weight, in order for the smaller to retain its value.

This problem arises in various parts of the PLS proof. For example, we would like the outputs of a circuit to be fed back to the inputs of the other one. The outputs have very small weight compared to the inputs, since the weights drop exponentially in the *Circuit Computing* gadget. We would like the inputs of Circuit 2 to change according to the outputs of circuit 1 and not the other way around. Another example involves the *Equality* Gadget, which influences the $Control_\ell$ of the *Circuit Computing* gadget. The nodes of the Gadget have weights of the order of 2^{10N}, while the control nodes of the *Circuit Computing* gadget are of the order of 2^{100N}. We would like the output of the gadget to bias the $Control_\ell$ nodes, while also remaining independent from them.

Let's get back to the original problem. A naive solution would be to connect node A directly with node B. However, this would result in node B biasing node A due to the larger weight it possesses. For example, if we connected *Control1* with the control variables of Circuit 2, then they would always bias *Control1* with a very large weight, rendering the entire *Equality* gadget useless. We would like to ensure that node A biases B with a relatively small weight, while also experiencing a small bias from it.

The solution we propose is a *Leveraging* gadget that is connected between nodes A and B. It's construction will depend on the weights A and B, as well as the bias that we would like B to experience from A. Before describing the construction, we discuss it's functionality on a high level.

As shown in Figure 4.15, we place the gadget between the nodes A and B. We use two parameters x, ϵ in the construction. We first want to ensure that node A experiences a small bias from the gadget. This is why we put nodes $L_{1,1}, L_{1,2}$ at the start with weight $B/2^{x+1} + \epsilon$, which puts a relatively small bias. We want these nodes to be dominated by A. This is why nodes $L_{1,3}, L_{1,4}$ have combined weight less than A. However, these nodes cannot directly influence B, since it's weight dominates the weights of $L_{1,1}, L_{1.2}$. For this reason, we repeat this construction $x + 1$ times, until nodes $L_{x,1}, L_{x,2}$, whose combined weight is slightly larger than B. This means that nodes $L_{x,3}, L_{x,4}$ are not dominated by B and can therefore be connected directly with it. The details of the proof are given below.

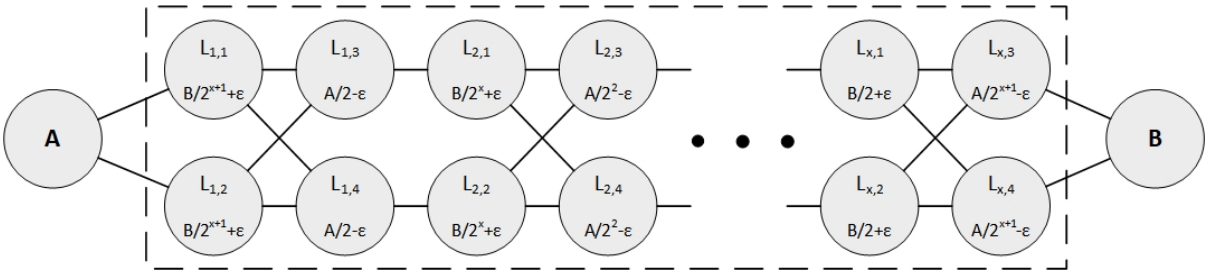

Figure 4.15: The Leveraging Gadget

Lemma 4.3.15. *If the input node A of a leverage gadget with output node B, parameters x, ϵ, has value 1, then the output node experiences bias $w_A/2^x + 2 * \epsilon$ towards 0, while the input node A experiences bias $w_B/2^x - 2 * \epsilon$ towards 1. If A has value 0, then B experiences the same bias towards 1, while A is biased towards 0.*

Proof. We first consider the nodes $L_{1,1}, L_{1,2}$. They both experience bias w_A towards the opposite value of A, which is greater than the remaining weight of their neighbors $2 * w_A - 2 * \epsilon$, and hence they are both dominated to take the opposite value of A. Similarly, the nodes $L_{1,3}, L_{1,4}$ are now biased to take the opposite values of $L_{2,1}, L_{2,2}$ with bias at least $w_B/2^x + 2 * \epsilon$, which is greater than the remaining neighbors of $w_B/2^x + \epsilon$. Hence, both $L_{1,3}, L_{1,4}$ have the same value as A in any equilibrium. In a similar way, we can prove that, in any equilibrium $L_{i,3} = L_{i,4} = A$, and therefore B experiences bias $w_A/2^x + 2 * \epsilon$ towards the opposite value of A, while A experiences bias at most $w_B/2^x - 2 * \epsilon$ from this gadget. ■

Note that the above lemma works for any value of ϵ. This means that we can make the bias that B experiences arbitrarily close to $w_B/2^x$. For all cases where such a *Leverage* gadget is used, it is implied that $\epsilon = 2^{-1000N}$ which is smaller than all other weights in the construction. Hence, we only explicitly specify the x parameter and, for simplicity, such a *Leverage* gadget is denoted as below schematically.

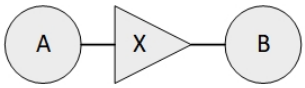

Figure 4.16: Leveraging Gadget *notation*

Chapter 5

Congestion Games

In this chapter we give a succinct introduction to the congestion game model. It is meant to serve both as a primer on the existing line of work, as well as provide motivation for our PLS results in the next section. Situations modeled by congestion games show up all around us on a daily basis. For example, imagine you have to cross the city center to get to a meeting you are late for, only it happens to be rush hour, and everyone is late for something and wishes to get there as fast as possible. In that case everyone will seek to take the shortest road (or the road they perceive to be the shortest!) which, for many people might coincide. Hence, many people will end up stuck in traffic. Situations like these are what's examined by the theory of congestion games, which asks questions such as: how much worse are situations like these made by the selfish behaviour of the drivers? does the traffic always reach a stable state in reasonable time? How hard is it to calculate that stable end state a priori?

5.1 Definitions

We begin by defining congestion games and showing some basic results about them.

Definition 5.1.1. A congestion game G is a tuple $G = \{N, F, (A_i)_{i\in[N]}, (d_f), f \in F\}$ where [N] denotes the set of players, F denotes the set of resources, $A_i \subset 2^F$ denotes the strategy space of player i and $d_f : N \to Z$ the cost function associated with resource f. $a = (a_1, ..., a_N)$ is an outcome of the game in which player i chooses strategy $a_i \in A_i$. For an outcome a we define the congestion $n_{f(a)}$ on resource f by $f(a) = |\{i \in [N] | f \in a_i\}|$. The cost for each player is the sum of the costs on the facilities of his strategy, $c_i(a) = \sum_{(f\in a_i)} d_f(n_f(a))$

In other words a congestion game is composed of resources that players can pick, which in turn cause the players to incur some cost based on the amount of users. Sometimes this set of resources can be a more meaningful combinatorial structure, like a network. In these cases players have paths as their allowable strategies and the resources are edges of a graph.

Another useful property that congestion games can often have is the following: the strategy sets of the players can coincide. In this case, we call the congestion game symmetric since everyone has access to the same choices as everyone else. In the network case it means that the game takes the form of an s-t routing game. Otherwise, the game is called asymmetric, or it is not specified. Moreover, if the players have different impacts on the delay functions among themselves then the game is called a weighted game, due to the presence of weights on players. Similar to all games, the most common behaviour is that of players acting selfishly given a state of the game. In this situation we consider the best-response dynamics of the game, which is the sequence of the states resulting from the iterative process of each player improving his lot. A state where the best response dynamics no longer induce any movement is called the Pure Nash Equilibrium, in the case of deterministic strategies, and the Mixed Nash Equilibrium in the case of mixed (randomized) strategies. Here we will be occupied principally with pure strategies and their properties.

The first question one seeks to answer is whether such a Pure Nash Equilibrium exists, and if so how can we calculate it? After all, mother nature seems to always find the equilibrium very quickly.

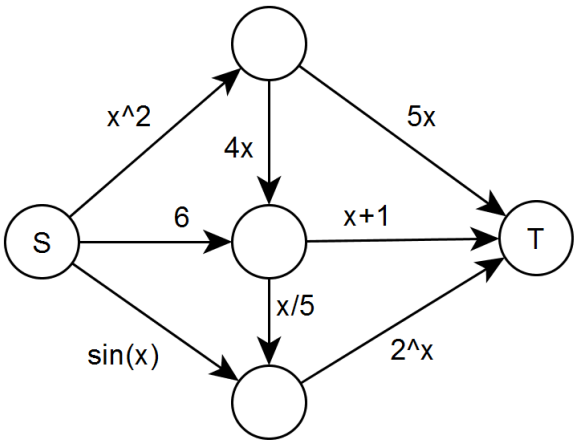

Figure 5.1: An example of a network congestion game

5.2 Potential functions

In certain situations local improvement dynamics have the property that they always converge to an equilibrium, i.e. after a finite amount of best-response moves we will be at a point where no one has incentive to change. This usually, happens because the transition graph of the game, where each node is a configuration of strategies, and edges correspond to best response moves, is directed and acyclic. In that case best-response moves will inevitably converge to some configuration and they will be unable to cycle. This is by no means the sole method of proving equilibria existence, the decision problem of determining existence of a PNE in congestion games has been well studied [Mil06],[Mil96].

There are generally two ways one uses to prove that the underlying transition graph has a DAG structure. The first way is the lexicographic method, wherein it is shown that configurations admit some sort of partially-ordered set structure. An example where this is useful is in the case of load scheduling, or in other words parallel links with weighted players, where we order the configurations by the satisfaction of each player from heaviest to lightest (i.e. lexicographic). The second, and easier, way is to use the potential method, which can be seen as a special case of the first approach. In the potential method we show that every configuration has some invariant that only gets increased with every best response move. This is called the potential function. The acyclic structure of a the transition graph follows from the existence of this function, since otherwise we would be able to have a directed cycle with strictly increasing values which is impossible.

A function F is a called an (exact) potential of a game G if it assigns a positive number to every configuration, such that:

$$F(S) - F(S'_i, S_{-i}) = c(S) - c(S'_i, S_{-i})$$

It turns out that such a function does exist for all (unweighted) congestion games. Discovered by [Ros73] in 1973, the following function does in fact capture the improvement dynamics of a congestion game.

Theorem 5.2.1. *$F(S) = \sum_{f \in F} \sum_{k=1}^{N_f} d_f(k)$ is an exact potential for unweighted congestion games, where N_f is the total load on resource f.*

Proof. $F(S) - F(S'_i, S_{-i}) = \sum_{f \in S_i \setminus S'_i} \sum_{k=1}^{N_f} d_f(k) + \sum_{f \in S'_i \setminus S_i} \sum_{k=1}^{N_f} d_f(k) - \sum_{f \in S_i \setminus S'_i} \sum_{k=1}^{N_f - 1} d_f(k) - \sum_{f \in S'_i \setminus S_i} \sum_{k=1}^{N_f + 1} d_f(k) = \sum_{f \in S_i \setminus S'_i} d_f(N_f) - \sum_{f \in S'_i \setminus S_i} d_f(N_f + 1) = \sum_{f \in S_i \setminus S'_i} d_f(N_f) + \sum_{f \in S_i \cap S'_i} d_f(N_f) - \sum_{f \in S'_i \cap S_i} d_f(N_f) - \sum_{f \in S'_i \setminus S_i} d_f(N_f + 1) = c(S) - c(S'_i, S_{-i})$ ■

Observe that since in every best response move a player moves from cost $c(S)$ to cost $c(S'_i, S_{-i})$ she experiences some finite improvement and therefore the potential gets reduced by some finite amount by every best response move. This immediately suggests a pseudopolynomial algorithm for computing a pure nash equilibrium of a congestion games. Indeed, repeating such a local optimization heuristic will inevitably return a fixed point. This is similar to what a Lyapunov function does in dynamical systems, only in this case the function is discrete. Unfortunately, there are counterexamples where this process can take a very long time to finish, simply because of tiny differences in the delay functions values. This can even happen with linear delays.

Hence, by the above function, we have that all congestion games are potential games. A significant question that arises from this is: Are there any potential games that aren't congestion games? The answer is, somewhat surprisingly, negative. In fact, it has been shown that every exact potential game can be cast in the form of a congestion game, in other words every potential game has an isomorphic congestion game. [MS96]. Hence, any and all inquiries into the nature and behaviour of potential games can be cast in terms of congestion games.

Theorem 5.2.2. *Every potential game is isomorphic to a congestion game.*

While this was originally proven in a rather complicated fashion by [MS96], there exists a more intuitive way of showing this by breaking the game into two subgames, due to [VBVM+99].

We begin by defining coordination and dummy games.

Definition 5.2.1. A strategic game $G = (N, S_{[N]}, c_{[N]})$ is:

- a coordination game if $c = c_i \forall i$ (i.e. everyone has the same goal)
- a dummy game if for all strategy pair S',S differing only in one player, $c(S_{-i}, S'_i) = c(S)$ (i.e. every player is indifferent to unilateral deviation

It turns out that a necessary and sufficient condition for a game to be a potential game is for it to be writable as the sum of a coordination and a dummy game.

Theorem 5.2.3. *A strategic game $G = (N, S_{[N]}, c_{[N]})$ is a potential game if and only if there exist functions c^d and c^c such that $c = c^d + c^c$ and $(N, S_{[N]}, c^d_{[N]})$, $(N, S_{[N]}, c^c_{[N]})$ are dummy and coordination games respectively.*

Proof. The if direction is obvious since we can use c^c as the potential.

For the "only if" direction let F be the potential function of the game. Also let $c^c = F$. Indeed, $(N, S_{[N]}, c^c_{[N]})$ then becomes a coordination game since F is the same for all players. It remains to prove that the residue, i.e. $c^d_i = c_i - F$ is a dummy game. Let us compute the change in a player's cost that results from a unilateral deviation. By the definition of the potential, $F(S_{-i}, S'_i) - F(S) = c(S_{-i}, S'_i) - c(S) = c(S_{-i}, S'_i) - c(S) - (c^d(S_{-i}, S'_i) - c^d(S)) \to 0 = c^d(S_{-i}, S'_i) - c^d(S)$ and hence c^d is a dummy game. ■

All that remains is to show that both dummy games and coordination games are isomorphic to a congestion games, separately. first, we define what it means for two game to be isomorphic. Informally, this means that the games are identical apart from possibly reordering their strategies.

Definition 5.2.2. Let two strategic games $G_1 = (N, S_{[N]}, c_{[N]})$, $G_2 = (N, S'_{[N]}, c'_{[N]})$ with identical player sets. They are said to be isomorphic if there is a bijection f such that for every configuration of player strategies we have that $c'(S'_1, .., S'_N) = c(f(S_1), .., f(S_N))$

Theorem 5.2.4. *Every coordination game is isomorphic to a congestion game.*

Proof. For every state of the game we define a resource $r(S)$, such that its cost is exactly $c(S)$ is all players are using it and 0 otherwise. The strategies of the players are $r(S_i, S{-}i))$ for all S_{-i}, i.e. every strategy of each player contains the resources corresponding to every configuration where he takes that particular strategy, with the other player ranging over the rest of the strategies. For example in a two-player three-strategy coordination game the strategy corresponding to strategy 1 in the coordination game would be $r(1,1), r(1,2), r(1,3)$. Note how every pair of players will always have a single common resource r(S), which will always be used by everyone and will be the only resource that incurs any delay. ■

Theorem 5.2.5. *Every dummy game is isomorphic to a congestion game.*

Proof. Essentially we desire a congestion game such that the delay of each player depends only on what resources the other players choose, regardless of his own choice. To do so we define one resource $r(S_{-i})$ for each player and for each configuration of the rest of the non-i players. The strategies for each player are: $r(S_{-i}) \cup r(x_j) | j \in N \setminus i, x_i \neq S_i$. Note that this means that every player has all the $r(S_{-i})$ resources but all of them except exactly one will be used by some other player. Indeed, the delay functions are setup in such a way that $c(r(S_{-i}))$ is equal to $c(S_{-i})$ when only one player is using it and 0 otherwise. ■

This concludes the proof of the isomorphism between potential games and congestion games, as we can pair up the strategies in the two separate congestion game instances we construct by taking their unions.

The above result cements the study of congestion games as central to understanding the behaviour of local optimization heuristic in the presence of a global potential function. Congestion games do not only constitute a useful model for a realistic situation but also confer insight into the nature of local optima and how hard they are to find.

5.3 Tractability of equilibria

As described in the previous section congestion games and potential games (lyapunov functions) are closely intertwined. The main object of interest in these situations is the actual minimum of this function, i.e. the nash equilibrium. It is, after all, the reason the existence of the potential is so important. In this section we are going to present several results concerning the question of finding that minimum. As described in the previous chapters, hardness guarantees are obtained in the form of PLS reductions, as well as the PSPACE completeness of the standard algorithm problem. The results below are organized in a hierarchical manner, where, in order of decreasing generality, we present the tractability status of the main point of interest of this thesis: the calculation of a PNE.

5.3.1 General congestion games

In the most general case where there is an arbitrary set of resources and strategies, [FPT04] devised a simple PLS completeness proof that is an immediate reduction from MaxCut. In particular, the congestion game they propose is composed of two strategies for each node, each of which contains one of two resources f_e, f'_e that correspond to each edge. These resources have cost w_e when used by zero or one player, but cost 0 when used by two players. Lastly, the strategy sets of each player are constrained to be the two strategies corresponding to each node. Note how this gives an immediate reduction from MaxCut considering that the chosen strategy for each player is the same as the side of the cut it would take. In the original paper, the reduction was from PosNae3Sat but the essence of the constructed instance is practically the same.

Theorem 5.3.1. *Computing a Pure Nash Equilibrium for general congestion games is PLS-complete.*

Note, however, that the above reduction only works for the general asymmetric case. What happens when one considers general congestion games again, only now the players have identical access to all strategies? Turns out that in this case as well the problem is PLS-complete, through a slight modification of the reduction in the previous paragraph. To modify the reduction above so that it can work in a symmetric strategic setting, one needs to somehow eliminate the possibility that players might all choose the same strategy, in which case the reduction would break down. The solution for this is simple. To every strategy set that we only want a single player to take part in, we add an exclusive resource which has a delay cost function with a huge jump. In particular, considering any asymmetric game's strategies S_i, we construct our symmetric game instance with strategies $S_i \cup e_i$, with $d_{e_i} = 0$ for at most one player and a very high number for more than one player. Clearly, in any equilibrium, since we have N strategies that each only fit one player and N players, the players will have separate strategies making the game indistinguishable from the asymmetric case.

Theorem 5.3.2. *Computing a Pure Nash Equilibrium for symmetric general congestion games is PLS-complete.*

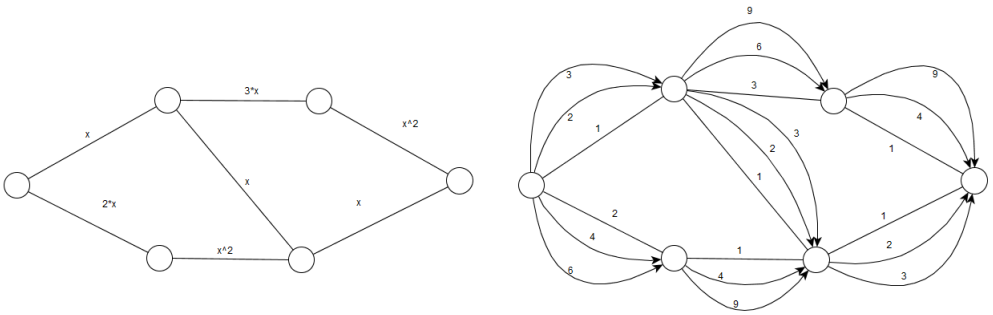

Figure 5.2: A symmetric network congestion game converted to a min-cost flow problem

What these results of [FPT04] tell us is that we will need to restrict ourselves further if we wish to have well behaved congestion games with tractable equilibria. General congestion games are simply too general and hard. The first such concession one makes is to consider congestion games rooted in reality, i.e. congestion games played on networks.

5.3.2 Network congestion games

In network congestion games the tractabilty of the pure equilibria has interesting distinctions depending on whether we consider the symmetric or the asymmetric case. In both case the Standard Algorithm Problem (i.e. best response) is PSPACE-hard, which means naive local search in general performs abysmally. There is in exception to that in case of series-parallel networks where best response does in fact work for efficient computation of a PNE [FKS05b]. We begin by presenting the positive results of [FPT04] for computing PNE using a simple flow algorithm for symmetric network games.

5.3.2.1 The symmetric case

Theorem 5.3.3 ([FPT04]). *There is a polynomial algorithm for finding a pure Nash equilibrium in symmetric network congestion games.*

The algorithm works by directly minimizing the Rosenthal potential instead of running a local search process. In particular, it creates the following network on which it runs a min-cost flow polynomial algorithm. Each edge of the congestion game network is replaced with N parallel edges, each of capacity 1 and each having cost $d_e(i)$. Then it is clear that any flow corresponds to a configuration of the congestion game whose Rosenthal potential value is equal to the flow being routed. Since such a flow can be globally minimized we immediately have a polynomial algorithm for global minimization of the potential, which also constitutes a nash equilibrium. This substitution process is illustrated in Figure 5.2.

This means that symmetric networks are easy to calculate equilibria for.

5.3.2.2 The asymmetric case

Unfortunately, the above algorithm cannot be extended to the network case where there are multiple sources and destinations. The main reason for that is because flows cannot be necessarily converted back to strategy configurations unambiguously. In fact, it can be shown that computation of equilibria in the asymmetric case is PLS-complete, and hence no algorithm can work, unless $PLS \subset FP$.

Theorem 5.3.4 ([FPT04],[ARV08]). *It is PLS-complete to compute any Pure Nash Equilibrium in asymmetric network congestion games.*

There are two ways to prove this result. The first way that this result was proven was by Fabrikant et al [FPT04]. Their proof consisted of reducing directly from CircuitFlip, and therefore was extraordinarily complex. In particular, their approach built on the MaxCut reduction of Yannakakis et al [Sch91] and added even more complications due to the fact that this is now a network game. However,

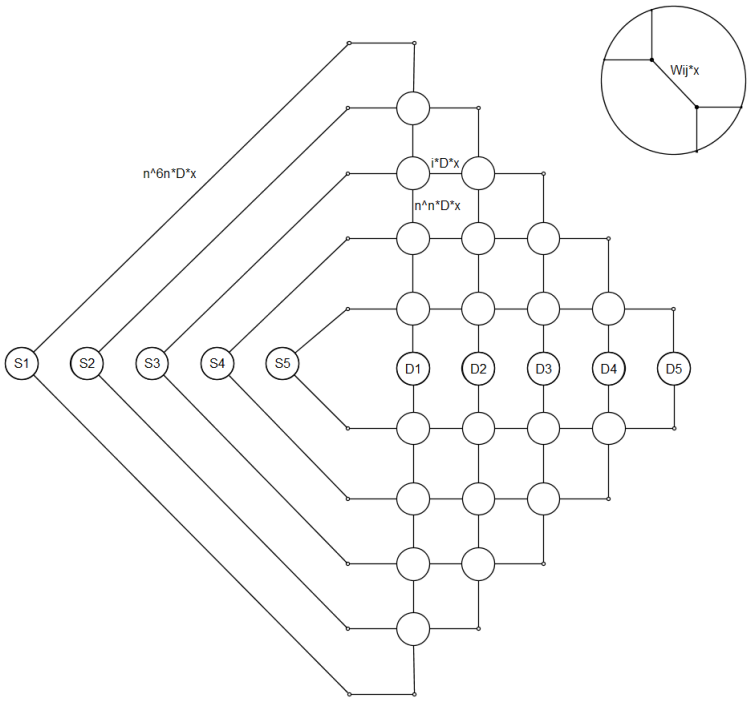

Figure 5.3: A small modification of the [ARV08] reduction

a few years later, Vocking et al [ARV08] produced an infinitely simpler PLS completeness proof for the same result, as an application of their PLS hardness framework for congestion games. We give here a general description of their proof. In their approach, the authors of [ARV08] define threshold games as an intermediate step for showing PLS complexity from MaxCut. While threshold games are a nice model that facilitates proving PLS-completeness for other problems and extensions, we find that the presentation of the [ARV08] construction is simpler when presented directly as a MaxCut to network games reduction. To be more specific, they construct a grid-like network as in Figure 5.3.

There are N players, each having as source the node S_i and destination the node D_i. The large diagonal edges have their delay functions setup in such a way that, regardless of what other players do, taking more than one such edge would be irrational, since it would lead to a strictly greater cost than necessary. Thus, players can only choose the above or the below segment of the network, corresponding to which side of the cut they take in the MaxCut instance. Lastly, once they choose a side they will incur exactly the cost of their neighbor who chose the same side, due to the way the intersections of this grid are set up. This completes the proof of the PLS-completeness of asymmetric (i.e. multi-commodity) network congestion games. We also note that this proof is tight in the sense of [Sch91], which means that the standard algorithm problem is PSPACE-hard, and any best response sequence has exponential length.

An interesting distinction can therefore be drawn between symmetric and asymmetric network games. Unlike the case of general abstract congestion games, there is an immense complexity gap between the tractability of the equilibria of these problems. Perhaps the same "symmetrization" technique that [FPT04] used to prove the hardness of symmetric games from asymmetric ones can be applied here as well? And if not, why?

The answer is that it can, only with a few caveats. Recall that this "symmetrization" technique simply boils down to adding a resource to each strategy so it is taken by exactly one player. In the case of networks this is tantamount to adding an edge to each source-target pair. Following this idea the symmetrized network would look like in Figure 5.4.

The reason this cannot give a reduction to the case of symmetric networks which is easily solvable through flows is that while one player takes one edge of the S edges, we have no guarantee it will end up taking the corresponding edge of the other side. For example a player might end up taking $S - S_1$ but also taking $D_4 - D$, which makes the hard instances of the grid-like networks of [ARV08] lose their

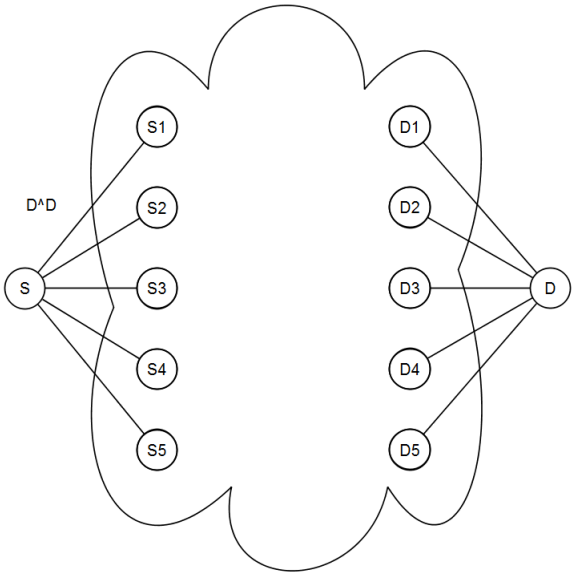

Figure 5.4: The "symmetrized" version of an asymmetric congestion network instance.

difficulty.

One thing that this approach can give us, on the other hand, is that if we started from an instance with players all taking their respective edges, i.e. player 1 taking $S - S_1$,$D_1 - D$ etc, then this property would be retained throughout any sort of best response movement. By recalling that the best response dynamics of asymmetric networks can perform exponentially badly, we have the following result.

Corollary 5.3.4.1 ([ARV08]). *For every N, there exists a symmetric network congestion game with N players, initial state S, polynomially bounded network size, and linear delay functions such that every best response sequence starting in S is exponentially long.*

The reason why this peculiar behaviour of symmetric networks to be both in FP, but also for their best response sequences to be PSPACE-hard, is interesting because it reveals that in many problems, even in PLS, local search isn't in fact the best approach! Another example of this is the simplex algorithm. It was shown by [FS15] that the simplex algorithm, for a specific pivot, (i.e. the local search heuristic for LPs) has PSPACE-hard properties. However, LPs have been known to be polynomially solvable ever since the ellipsoid method, or any of the interior point algorithms came into public view. It is worth mentioning here one of the very few results that concern computing Nash equilibria without using local search. Specifically, in [GLMM10] the authors design an algorithm based on maximum flows that computes an equilibrium state in a restricted links game.

As we saw in the last two sections, local improvement sequences both in the case of symmetric and asymmetric networks can perform arbitrarily badly. Despite that, there exist specific cases where best-response can in fact be a legitimate tactic. We briefly present two such subcases below.

5.3.2.3 Series-Parallel Networks

Series parallel graphs consist of a very simple topology of edges connected either in parallel or in sequence, repeated recursively. More rigorously:

Definition 5.3.1. A Series-Parallel graph is either:

- an edge
- two series-parallel graphs connected in parallel
- two series-parallel graphs connected in series

Despite their simple appearance, such graphs have an interesting necessary and sufficient condition for identifying them. Namely, [BBT85] showed that a network is series parallel if and only if a greedy algorithm can calculate a maximum flow. This interesting property was later leveraged by [FKS05b] to show that a best response algorithm does in fact converge quickly in the case of series parallel networks (and effectively only in those). This is because once a player settles into his best response, then no matter how the others change he will have no incentive to change his choice, and thus every player will move at most once.

Hence, in this simple case of symmetric networks best response is a valid strategy for computing a Nash Equilibrium.

5.3.2.4 Matroid Games

Another type of game where best response is quite efficient is the case of congestion games where the resources have some matroid property.

Definition 5.3.2. A tuple $M = (R, I)$ is a matroid if R is a finite set of resources and I is a nonempty family of subsets of R such that if $I \in I$ and $J \subset I$, then $J \in I$, and if $I, J \in I$ and $|J| < |I|$, then there exists an $i \in I \setminus J$ with $J \cup \{i\} \in I$

Essentially, the matroid combinatorial structure embodies settings where we have some exchange property, similar to bases in linear algebra.

We are not going to go much into detail regarding these games, but we are going to note that in the general case such games are MST allocation games (recall that the set of Minimum Spanning Trees satisifes an exchange property and is discrete convex), but in the case of matroid congestion games they devolve into a bunch of parallel links connected in parallel, which isnt a very interesting topology.

Nevertheless, a matroid conestion game can have its best response sequences converge in $O(n^2 * m * rank)$ moves as shown in the seminal paper of Vocking et al [ARV08].

5.3.3 Approximate equilibria

While we often seek exact equilibria where all players have no incentive to change, they are often quite intractable as we saw in the previous sections. For this reason, an alternate definition of equilibrium that has been well-examined is the notion of an approximate equilibrium. In an approximate equilibrium players cannot unilaterally deviate from their choice and increase their payoff more than a certain factor. In particular, there is no configuration u and player i such that $c(u_{-i}, u_i') \leq a * c(u)$. The main intellectual attraction of such approximate equilibria is several results on their tractability. One of the most important such results concerns an algorithm by Caragiannis et al for computation of approximate equilibria in congestion games [CFGS11]. This algorithm has also been extended for approximate equilibria in weighted games [CF19]. Despite its power, the aforementioned algorithm has the drawback on only being applicable to specific delay functions, and not general functions, which can provably have PLS-hard equilibria [SV08], and hence exponentially long best-response chains. Moreover, there have been significant advancements in the case that the players seek to optimize their configuration towards an optimal social value or , even though these are not approximate equilibria. [FM09],[LMM03]m[CS11],[AAE+08].

5.4 Weighted congestion games

As we saw in the previous sections, it is clear that the "lay of the land" concerning the tractability of congestion games, both symmetric and asymmetric is well examined. However, congestion games can admit an interesting generalization in the form of games where the players are allowed to control nonuniform atomic units of flow, i.e. they are weighted.

These games were introduced by Fotakis et al in [FKS05a], where it was shown that for an equilibrium to exist a necessary condition is for the delay functions to be affine. In that case, a potential function can be constructed, and similar to vanilla congestion games, a guarantee for the eventual convergence of nash best response dynamics obtained. The difference in this case is that the differential of the potential function due to a player's best-response move is not exactly the value he improves his lot with, but instead his improvement multiplied by his weight.

Theorem 5.4.1. $\sum_{i \in N} w_i \sum_{e \in s_i} (a_e * w_i + b_e) + \sum_{e \in E} s_e * (a_e * s_e + b_e)$ *is a weighted potential function for weighted congestion games with affine delays.*

We note that affine function arent the only family of delays that admits a weighted potential. As shown in [PS07] similar results can be obtained for networks with exponential functions. As [HKM11] show these are the only two function classes admitting weighted equilibria. It is possible by defining the impact of weighted players one the costs in a more sophisticated way to obtain more games with existing equilibria. [KR15].

In constrast with unweighted games, however the complexity of equilibria has not been well studied. All results regarding the tractability of Nash equilibria in weighted congestion games are so far directly derived as edge cases of unweighted games. For example, we know that weighted congestion games with affine functions are PLS-complete, but only because we know that unweighted games with affine functions are PLS-complete.

The motivating question of this work is to examine the impact of weights on the complexity of the nash equilibria. Therefore, we focus our attention on case of congestion games where the unweighted variants are in FP. The first such problem concerns symmetric networks, and in fact a very specific type of them which was mentioned earlier, which are series-parallel weighted congestion games. Note that in [FKS05b] weighted congestion games in a series parallel setting was proven to not be amenable to best response dynamics, but the question of its PLS-completeness was left unanswered.

Theorem 5.4.2. *Series-parallel weighted congestion games are PLS-complete.*

The proof of this theorem utilizes a direct reduction from MaxCut and is presented in the next section.

The major technical work of this thesis, i.e. the PLS-completeness of node weighted maximal cut, was motivated by the assumption of using identity delays in a congestion game. While it has its own independent interest, its main use to us is showing PLS-completeness even in games where the edges are unweighted, but only players have weights. Hence we obtain the following novel result about weighted congestion games with identity delays.

Theorem 5.4.3. *Asymmetric weighted congestion games with identity delay functions are PLS-complete.*

Again, the proof of this is presented in the final chapter.

Overall, this clears up significantly the landscape regarding complexity of equilibria in the weighted settings specifically. A central question remains, however, which pertains to the complexity of equilibria when a game is both symmetric and has identity delay functions.

Chapter 6

PLS completeness of Weighted Congestion Games

In this section we state and provide proofs for our PLS-completeness theorems for congestion games. In the first part we do so for single-commodity Weighted Network Congestion Games on series parallel networks with linear latencies (Theorem 6.1.1) and in the second part we do so for multi-commodity Weighted Network Congestion Games with the identity function on all links (Theorem 6.2.1), by applying the result of the section devoted to NodeMaxCut.

6.1 Series-parallel weighted congestion games

We prove that it is PLS-hard to compute an equilibrium of a Weighted Congestion Game, even if it is a Weighted Network Congestion Game on a series-parallel single-commodity network with linear latencies, i.e., with latency functions of the form $\ell_k(x) = a_k x$. We do so by reducing from MaxCut.

Theorem 6.1.1. *Computing a Nash equilibrium in single-commodity Weighted Network Congestion Games with linear latency functions is PLS-complete.*

Proof. We will reduce from the PLS-complete problem MaxCut. Given an instance of MaxCut we will construct a Weighted Network Congestion Game for which the Nash equilibria will correspond to maximal solutions of MaxCut and vice versa.

To give the construction, let $H(N, A)$ be an edge-weighted graph of a MaxCut instance and let $n = |N|$ and $m = |A|$. In the constructed Weighted Network Congestion Game instance there will be $3n$ players which will share n different weights inside the set $\{16^i : i \in [n]\}$ so that for every $i \in [n]$ there are exactly 3 players having weight $w_i = 16^i$. All players share a common origin-destination pair $o - d$ and choose $o - d$ paths on a series-parallel graph G. Graph G is a parallel composition of two identical copies of a series-parallel graph.We call these copies G_1 and G_2. In turn, each of G_1 and G_2 is a series composition of m different series-parallel graphs, each of which corresponds to the m edges of H. For every $\{i, j\} \in A$ let F_{ij} be the series-parallel graph that corresponds to edge $\{i, j\}$. F_{ij} is presented in Fig. 6.1, where D is assumed to be a (polynomially) big enough constant. An example graph G is given in Fig. 6.2.

Observe that in each of G_1 and G_2 there is a unique path that contains all the links with latency functions $\ell_i(x)$, for $i \in [n]$, and call these paths p_i^u and p_i^l for the upper (G_1) and lower (G_2) copy respectively. Note that each of p_i^u and p_i^l in addition to those links, contains some links with latency function of the form $\frac{w_{ij}x}{w_i w_j}$. These links for path p_i^u or p_i^l are in one to one correspondence to the edges of node i in H and this is crucial for the proof.

By the choice of the players' weights and the latency functions' slopes, one can show that at a Nash equilibrium, a player of weight w_i chooses either p_i^u or p_i^l. The formal proof uses induction starting from larger weights. The heaviest players, i.e., players with weight w_n, have a dominant strategy to choose either p_n^u or p_n^l since $\ell_n(x)$ has a significantly smaller slope than all other $\ell_i(x)$'s, small enough so that even if all other players choose the same paths (reaching a load of at most $3\sum_{l=1}^{n} 16^l = \frac{16^{n+1}-1}{16-1}$), still

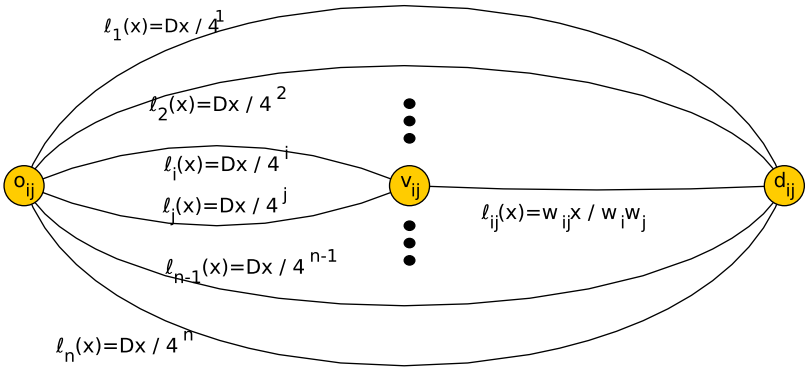

Figure 6.1: The series-parallel network F_{ij} that corresponds to edge $\{i,j\} \in A$

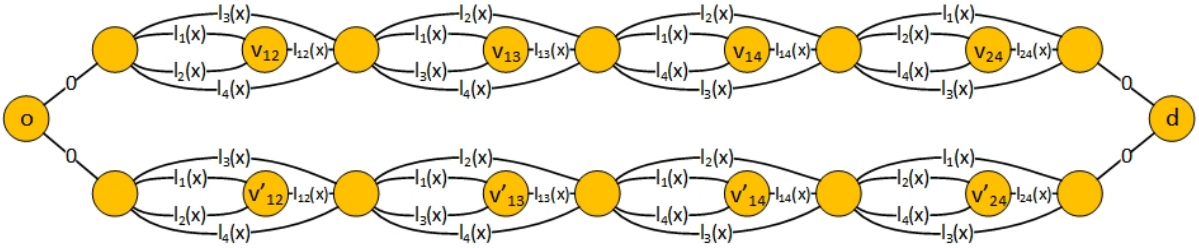

Figure 6.2: An example of the structure of the construction for a graph $H(N,A)$, with $N = \{1,2,3,4\}$ and $A = \{\{1,2\},\{1,3\},\{1,4\},\{2,4\}\}$. Each of the upper and lower parts consists of copies of F_{12}, F_{13}, F_{14} and F_{24} connected in series.

players of weight w_n prefer p_n^u or p_n^l over all other paths. But then, p_n^u and p_n^l get a lot of weight load at equilibrium compared to the weight of lighter players. This makes the links on these paths look like they get some big additive constants, which makes them extremely expensive for all lighter players and these players exclude them from their strategy space. That said, by the same reasoning, the players of weight w_{n-1} have a dominant strategy to choose either p_{n-1}^u or p_{n-1}^l and this inductively proves true for all $i \in [n]$.

Additionally, one can prove that p_i^u and p_i^l will have at least one player (of weight w_i). The underlying idea is that if wlog p_i^u has two players (of weight w_i) then the third player of weight w_i prefers to go to p_i^l, since it is going to be empty. This already provides a good structure of a Nash equilibrium and players of different weights, say w_i and w_j, may go through the same link in G (the edge with latency function $w_{ij}x/w_iw_j$) only if $\{i,j\} \in A$. The correctness of the reduction lies in the fact that players in G try to minimize their costs incurred by these type of links in the same way one wants to minimize the sum of the weights of the edges in each side of the cut when solving LocalMaxCut.

Given a maximal solution S of MaxCut the proof shows that the configuration Q that for every $k \in S$ routes 2 players through p_k^u and 1 player through p_k^l and for every $k \in N \setminus S$ routes 1 player through p_k^u and 2 players through p_k^l is an equilibrium. Conversely, given an equilibrium Q the cut $S = \{k \in N : 2 \text{ players have chosen } p_k^u \text{ at } Q\}$ is a maximal solution of LocalMaxCut.

Assume that we are at equilibrium and consider a player of weight w_k that has chosen p_k^u and wlog p_k^u is chosen by two players (of weight w_k). By the equilibrium conditions the cost she computes for p_k^u is at most the cost she computes for p_k^l, which implies

$$\sum_{i=1}^{m} \frac{2D16^k}{4^k} + \sum_{\{k,j\}\in A} \frac{w_{kj}(2\cdot 16^k + x_j^u 16^j)}{16^k 16^j} \leq \sum_{i=1}^{m} \frac{2D16^k}{4^k} + \sum_{\{k,j\}\in A} \frac{w_{kj}(2\cdot 16^k + x_j^l 16^j)}{16^k 16^j}$$

where x_j^u (resp. x_j^l) is either 1 or 2 (resp. 2 or 1) depending whether, for any $j : \{k,j\} \in A$, one or two

players (of weight w_j) respectively have chosen path p_j^u. By canceling out terms, the above implies

$$\sum_{\{k,j\}\in A} w_{kj}x_j^u \leq \sum_{\{k,j\}\in A} w_{kj}x_j^l \Leftrightarrow \sum_{\{k,j\}\in A} w_{kj}(x_j^u - 1) \leq \sum_{\{k,j\}\in A} w_{kj}(x_j^l - 1) \tag{6.1}$$

Define $S = \{i \in N : x_i^u = 2\}$. By our assumption it is $k \in S$ and the left side of (6.1), i.e., $\sum_{\{k,j\}\in A} w_{kj}(x_j^u - 1)$, is the sum of the weights of the edges of H with one of its nodes being k and the other belonging in S. Similarly, the right side of of (6.1), i.e., $\sum_{\{k,j\}\in A} w_{kj}(x_j^l - 1)$ is the sum of the weights of the edges with one of its nodes being k and the other belonging in $N \setminus S$. But then (6.1) directly implies that for the (neighboring) cut S' where k goes from S to $N \setminus S$ it holds $W(S) \geq W(S')$. Since k was arbitrary (given the symmetry of the problem), this holds for every $k \in [n]$ and thus for every $S' \in ND(S)$ it is $W(S) \geq W(S')$ proving one direction of the claim. Observing that the argument works backwards the proof completes. ■

6.2 Multi-Commodity Weighted Network Congestion Games with identity delays

We prove that it is PLS-hard to compute an equilibrium in Weighted Congestion Games, even if it is a Weighted Network Congestion Game with all latency functions equal to the identity function, i.e., for any link e, $\ell_e(x) = x$. This result is stronger in some aspect than that of Section 6.1 since we allow only the weights of the players to be exponential. Note that if both the coefficients of the linear latency functions and the weights of the players are polynomial, then best response dynamics converges to an equilibrium in polynomial time. For the proof, we reduce from NodeMaxCut which, as we prove in Theorem 4.2.11, is PLS-complete.

Theorem 6.2.1. *Computing a Nash equilibrium in multi-commodity Weighted Network Congestion Games with all links having the identity function as their latency function is PLS-complete.*

Proof. We will reduce from the PLS-complete problem NodeMaxCut. Our construction draws ideas from Ackermann et al. [ARV08]. For an instance of NodeMaxCut we will construct a multi-commodity Network Congestion Game where every equilibrium will correspond to a maximal solution of NodeMaxCut and vice versa. For the formal PLS-reduction, which needs functions ϕ_1 and ϕ_2, ϕ_1 returns the (polynomially) constructed instance described below and ϕ_2 will be revealed later in the proof.

We will use only the identity function as the latency function of every link, but for ease of presentation we will first prove our claim assuming we can use constant latency functions on the links. Then we will describe how we can drop this assumption and use only the identity function on all links, and have the proof still going through.

Let $H(N, A)$ be the node-weighted graph of an instance of NodeMaxCut and let $n = |N|$ and $m = |A|$. The Weighted Network Congestion Game will have n players, with player i having her own origin destination $o_i - d_i$ pair and weight w_i equal to the weight of node $i \in N$. In the constructed network there will be many $o_i - d_i$ paths for every player i but there will be exactly two paths that cost-wise dominate all others. At equilibrium, every player will choose one of these two paths that correspond to her. This choice for player i will be equivalent to picking the side of the cut that node i should lie in order to get a maximal solution of NodeMaxCut.

The initial network construction is shown in Fig. 6.3. It has n origins and n destinations. The rest of the vertices lie either on the lower-left half (including the diagonal) of a $n \times n$ grid, which we call the upper part, or the upper-left half of another $n \times n$ grid, which we call the lower part. Other than the links of the two half-grids that are all present, there are links connecting the origins and the destinations to the two parts. For $i \in [n]$, origin o_i in each of the upper and lower parts connects to the first (from left to right) vertex of the row that has i vertices in total. For $i \in [n]$, destination d_i in each of the upper and lower parts connects to the i-th vertex of the row that has n vertices in total. To define the (constant) latency functions, we will need 2 big constants, say d and $D = n^3 d$, and note that $D \gg d$.

All links that connect to an origin or a destination and all the vertical links of the half-grids will have constant D as their latency function, and any horizontal link that lies on a row with i vertices will

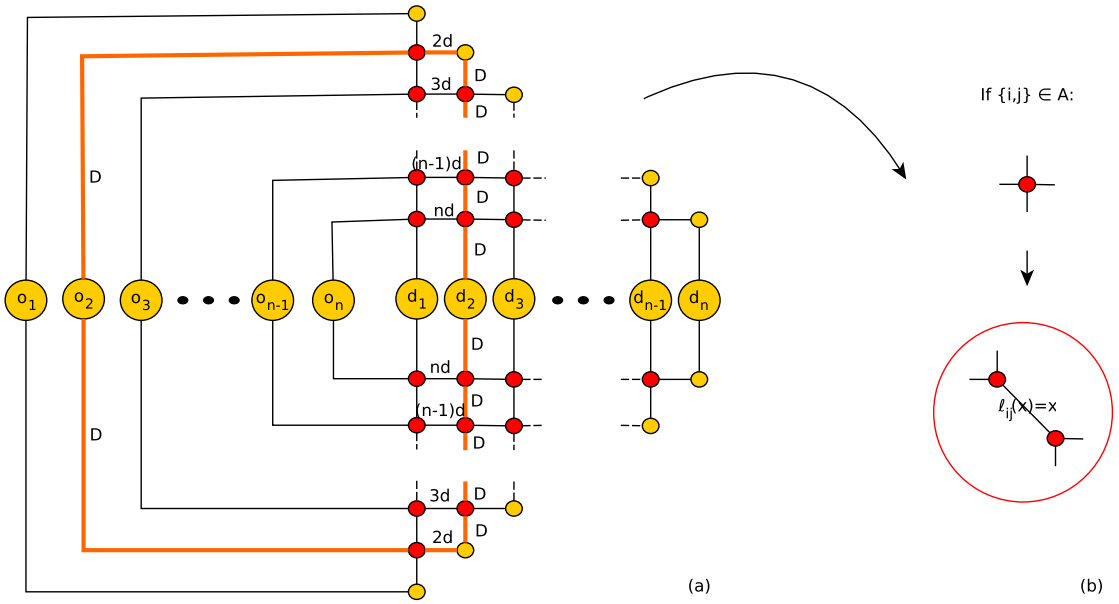

Figure 6.3: (a) The construction of the reduction of Theorem 6.2.1. As an example, in orange are the least costly $o_2 - d_2$ paths p_2^u (up) and p_2^l (down), each with cost equal to $2D + 2d + (n-2)D$. (b) The replacement of the red node at the i-th row and j-th column of the upper half-grid whenever edge $\{i,j\} \in A$. A symmetric replacement happens in the lower half-grid.

have constant $i \cdot d$ as its latency function. To finalize the construction we will do some small changes but note that, as it is now, player i has two shortest paths that are far less costly (at least by d) than all other paths. These two paths are path p_i^u that starts at o_i, continues horizontally through the upper part for as much as it can and then continues vertically to reach d_i, and path p_i^l which does the exact same thing through the lower part (for an example see Fig. 6.3a). Each of p_i^u and p_i^l costs equal to $c_i = 2D + i(i-1)d + (n-i)D$. To verify this claim simply note that (i) if a path tries to go through another origin or moves vertically away from d_i in order to reach less costly horizontal links, then it will have to pass through at least $(2+i-1)+2$ vertical links of cost D and its cost from such edges compared to p_i^u's and p_i^l's costs increases by at least $2D = 2n^3 d$, which is already more than paying all horizontal links; and (ii) if it moves vertically towards d_i earlier than p_i^u or p_i^l then its cost increases by at least d, since it moves towards more costly horizontal links.

To complete the construction if $\{i,j\} \in A$ (with wlog $i < j$) we replace the (red) vertex at position i,j of the upper and the lower half-grid ($1,1$ is top left for the upper half-grid and lower left for the lower half-grid) with two vertices connected with a link, say e_{ij}^u and e_{ij}^l respectively, with latency function $\ell_{ij}(x) = x$, where the first vertex connects with the vertices at positions $i, j-1$ and $i-1, j$ of the grid and the second vertex connects to the vertices at positions $i+1, j$ and $i, j+1$ (see also Fig. 6.3b). Note that if we take $d \gg \sum_{k\in[n]} w_k$, then, for any $i \in [n]$, paths p_i^u and p_i^l still have significantly lower costs than all other $o_i - d_i$ paths. Additionally, if $\{i,j\} \in A$ then p_i^u and p_j^u have a single common link and p_i^l and p_j^l have a single common link, namely e_{ij}^u and e_{ij}^l respectively, which add some extra cost to the paths (added to c_i defined above).

Assume we are at equilibrium. By the above discussion player $i \in [n]$ may only have chosen p_i^u or p_i^l. Let $S = \{i \in [n] : \text{player } i \text{ has chosen } p_i^u\}$. We will prove that S is a solution to NodeMaxCut. By the equilibrium conditions for every $i \in S$ the cost of p_i^u, say c_i^u, is less than or equal to the cost of p_i^l, say c_i^l. Given the choices of the rest of the players, and by defining S_i to be the neighbors of i in S, i.e. $S_i = \{j \in S : \{i,j\} \in A\}$, and N_i be the neighbors of i in N, i.e. $N_i = \{j \in N : \{i,j\} \in A\}$, $c_i^u \le c_i^l$ translates to

$$\Big(2D + i(i-1)d + (n-i)D\Big) + \Big(\sum_{j\in N_i} w_i + \sum_{j\in S_i} w_j\Big) \le \Big(2D + i(i-1)d + (n-i)D\Big) + \Big(\sum_{j\in N_i} w_i + \sum_{j\in N_i\setminus S_i} w_j\Big)$$

with the costs in the second and fourth parenthesis coming from the e_{ij}'s for the different j's. This equivalently gives

$$\sum_{j \in S_i} w_j \leq \sum_{j \in N_i \setminus S_i} w_j \Leftrightarrow \sum_{j \in S_i} w_i w_j \leq \sum_{j \in N_i \setminus S_i} w_i w_j.$$

The right side of the last inequality equals to the weight of the edges with i as an endpoint that cross cut S. The left side equals to the weight of the edges with i as an endpoint that cross the cut S', where S' is obtained by moving i from S to $N \setminus S$. Thus for S and S' it is $W(S') \leq W(S)$. A similar argument (or just symmetry) shows that if $i \in N \setminus S$ and we send i from $N \setminus S$ to S to form a cut S' it would again be $W(S') \leq W(S)$. Thus, for any $S' \in ND(S)$ it is $W(S) \geq W(S')$ showing that S is a solution to NodeMaxCut. Observing that the argument works backwards we have that from an arbitrary solution of NodeMaxCut we may get an equilibrium for the constructed Weighted Congestion Game instance. For the formal part, to define function ϕ_2, given the constructed instance and one of its solutions, ϕ_2 returns solution $s = \{i \in [n] : \text{player } i \text{ has chosen } p_i^u\}$.

What remains to show is how we can almost simulate the constant latency functions so that we use only the identity function on all links and, for every $i \in [n]$, player i still may only choose paths p_i^u or p_i^l at equilibrium. Observe that, since we have a multi-commodity instance we can simulate (exponentially large) constants by replacing a link $\{j, k\}$ with a three link path $j - o_{jk} - d_{jk} - k$, adding a player with origin o_{jk} and destination d_{jk} and weight equal to the desired constant. Depending on the rest of the structure we may additionally have to make sure (by suitably defining latency functions) that this player prefers going through link $\{o_{jk}, d_{jk}\}$ at equilibrium.

To begin with, consider any horizontal link $\{j, k\}$ with latency function $i \cdot d$ (for some $i \in [n]$) and replace it with a three link path $j - o_{jk} - d_{jk} - k$. Add a player with origin o_{jk} and destination d_{jk} with weight equal to in^3w, where $w = \sum_{i \in [n]} w_i$, and let all links have the identity function. At equilibrium no matter the sum of the weights of the players that choose this three link path, the $o_{jk} - d_{jk}$ player prefers to use the direct $o_{jk} - d_{jk}$ link or else she pays at least double the cost (middle link vs first and third links). Thus the above replacement is (at equilibrium) equivalent to having link $\{j, k\}$ with latency function $3x + in^3w = 3x + i \cdot d$, for $d = n^3w$.

Similarly, consider any link $\{j, k\}$ with latency function D and replace it with a three link path $j - o_{jk} - d_{jk} - k$. Add a player with origin o_{jk} and destination d_{jk} with weight equal to n^3d and let all links have the identity function. Similar to above, this replacement is (at equilibrium) equivalent to having link $\{j, k\}$ with latency function $3x + n^3d = 3x + D$, for $D = n^3d$.

With these definitions, at equilibrium, all complementary players will go through the correct links and, due to the complementary players, all links that connect to an origin or a destination will have cost $\approx D$, all vertical links of the half-grids will cost $\approx D$, and any horizontal link that lies on a row with i vertices will cost $\approx i \cdot d$, where "$\approx$" means at most within $\pm 3w = \pm \frac{3d}{n^3}$ (note that w is the maximum weight that the $o_i - d_i$ players can add to each of the three link paths). Additionally, for every $i \in [n]$, p_i^u and p_i^l are structurally identical, i.e., they have the same structure, identical complementary players on their links and share the same latency functions. All the above make the analysis go through in the same way as in the simplified construction. ■

Bibliography

[AAE+08] Baruch Awerbuch, Yossi Azar, Amir Epstein, Vahab Seyed Mirrokni, and Alexander Skopalik. Fast convergence to nearly optimal solutions in potential games. In *Proceedings of the 9th ACM conference on Electronic commerce*, pages 264–273. ACM, 2008.

[AAL03] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[Aar06] Scott Aaronson. Lower bounds for local search by quantum arguments. *SIAM Journal on Computing*, 35(4):804–824, 2006.

[ABPW17] Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 429–437. ACM, 2017.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[AMR11] David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the k-means method. *Journal of the ACM (JACM)*, 58(5):19, 2011.

[ARV08] Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *Journal of the ACM (JACM)*, 55(6):25, 2008.

[BBT85] Wolfgang W Bein, Peter Brucker, and Arie Tamir. Minimum cost flow algorithms for series-parallel networks. *Discrete Applied Mathematics*, 10(2):117–124, 1985.

[BG94] Mihir Bellare and Shafi Goldwasser. The complexity of decision versus search. *SIAM Journal on Computing*, 23(1):97–119, 1994.

[CF19] Ioannis Caragiannis and Angelo Fanelli. On approximate pure nash equilibria in weighted congestion games with polynomial latencies. *arXiv preprint arXiv:1902.07173*, 2019.

[CFGS11] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient computation of approximate pure nash equilibria in congestion games. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 532–541. IEEE, 2011.

[Cro58] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[CS11] Steve Chien and Alistair Sinclair. Convergence to approximate nash equilibria in congestion games. *Games and Economic Behavior*, 71(2):315–327, 2011.

[DM13] Dominic Dumrauf and Burkhard Monien. On the pls-complexity of maximum constraint assignment. *Theoretical Computer Science*, 469:24–52, 2013.

[DMT09] Dominic Dumrauf, Burkhard Monien, and Karsten Tiemann. Multiprocessor scheduling is pls-complete. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2009.

[DP11] Constantinos Daskalakis and Christos Papadimitriou. Continuous local search. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 790–804. SIAM, 2011.

[ET11] Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In *International Colloquium on Automata, Languages, and Programming*, pages 171–182. Springer, 2011.

[FKS05a] Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2-3):226–239, 2005.

[FKS05b] Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. Symmetry in network congestion games: Pure equilibria and anarchy cost. In *International Workshop on Approximation and Online Algorithms*, pages 161–175. Springer, 2005.

[FM09] Angelo Fanelli and Luca Moscardelli. On best response dynamics in weighted congestion games with polynomial delays. In *International Workshop on Internet and Network Economics*, pages 55–66. Springer, 2009.

[FPT04] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 604–612. ACM, 2004.

[FRG17] Aris Filos-Ratsikas and Paul W Goldberg. Consensus halving is ppa-complete. *arXiv preprint arXiv:1711.04503*, 2017.

[FS15] John Fearnley and Rahul Savani. The complexity of the simplex method. In *Proceedings of the forty-seventh annual ACM symposium on theory of computing*, pages 201–208. ACM, 2015.

[GLMM10] Martin Gairing, Thomas Lücking, Marios Mavronicolas, and Burkhard Monien. Computing nash equilibria for scheduling on restricted parallel links. *Theory of Computing Systems*, 47(2):405–432, 2010.

[GP17] Paul W Goldberg and Christos H Papadimitriou. Tfnp: an update. In *International Conference on Algorithms and Complexity*, pages 3–9. Springer, 2017.

[GS10] Martin Gairing and Rahul Savani. Computing stable outcomes in hedonic games. In *International Symposium on Algorithmic Game Theory*, pages 174–185. Springer, 2010.

[HKM11] Tobias Harks, Max Klimm, and Rolf H Möhring. Characterizing the existence of potential functions in weighted congestion games. *Theory of Computing Systems*, 49(1):46–70, 2011.

[HY17] Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1352–1371. SIAM, 2017.

[Jeř16] Emil Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016.

[JPY88] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

[K+41] Shizuo Kakutani et al. A generalization of brouwer's fixed point theorem. *Duke mathematical journal*, 8(3):457–459, 1941.

[Kaz19] Artem Kaznatcheev. Computational complexity as an ultimate constraint on evolution. *Genetics*, 212(1):245–265, 2019.

[KR15] Konstantinos Kollias and Tim Roughgarden. Restoring pure equilibria to weighted congestion games. *ACM Transactions on Economics and Computation (TEAC)*, 3(4):21, 2015.

[Kre89] Mark W Krentel. Structure in locally optimal solutions. In *30th Annual Symposium on Foundations of Computer Science*, pages 216–221. IEEE, 1989.

[LMM03] Richard J Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of the 4th ACM conference on Electronic commerce*, pages 36–41. ACM, 2003.

[MAK07] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical aspects of local search*. Springer Science & Business Media, 2007.

[Mil96] Igal Milchtaich. Congestion games with player-specific payoff functions. *Games and economic behavior*, 13(1):111–124, 1996.

[Mil06] Igal Milchtaich. The equilibrium existence problem in finite network congestion games. In *International Workshop on Internet and Network Economics*, pages 87–98. Springer, 2006.

[MP91] Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[MS96] Dov Monderer and Lloyd S Shapley. Potential games. *Games and economic behavior*, 14(1):124–143, 1996.

[MT10] Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

[Pap92] Christos H Papadimitriou. The complexity of the lin–kernighan heuristic for the traveling salesman problem. *SIAM Journal on Computing*, 21(3):450–465, 1992.

[Pap94] Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994.

[PS07] Panagiota N Panagopoulou and Paul G Spirakis. Algorithms for pure nash equilibria in weighted congestion games. *Journal of Experimental Algorithmics (JEA)*, 11:2–7, 2007.

[Ros73] Robert W Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[Sch91] Alejandro A Schäffer. Simple local search problems that are hard to solve. *SIAM journal on Computing*, 20(1):56–87, 1991.

[ST04] Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.

[SV08] Alexander Skopalik and Berthold Vöcking. Inapproximability of pure nash equilibria. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 355–364. ACM, 2008.

[SZZ18] Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. Ppp-completeness with connections to cryptography. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 148–158. IEEE, 2018.

[VBVM+99] Mark Voorneveld, Peter Borm, Freek Van Megen, Stef Tijs, and Giovanni Facchini. Congestion games and potentials reconsidered. *International Game Theory Review*, 1(03n04):283–299, 1999.