From static code distribution to more shrinkage for the multiterminal cut

Bram De Wachter*, Alexandre Genon*, and Thierry Massart

Université Libre de Bruxelles
Département d'Informatique, Bld du Triomphe, B-1050 Bruxelles
{bdewacht,agenon,tmassart}@ulb.ac.be

Abstract. We present the problem of statically distributing instructions of a common programming language, a problem which we prove equivalent to the multiterminal cut problem. We design efficient shrinkage techniques which allow to reduce the size of an instance in such a way that optimal solutions are preserved. We design and evaluate a fast local heuristics that yields remarkably good results compared to a well known $2 - \frac{2}{k}$ approximation algorithm. The use of the shrinkage criterion allows us to increase the size of the instances solved exactly or augments the precision of any particular heuristics.

1 Introduction

We present the problem of automatic distribution of programming languages, motivated by our research in automatic distributed industrial control systems [14]. This problem consists in distributing a program code among different sites, minimizing the total communications between these sites during its execution. We show that this problem is NP-hard. Furthermore, we show that it is equivalent to the *multiterminal cut* presented in [5], and will therefore concentrate on finding new ways to attack the problem described in terms of multiterminal cut.

The key concept used in this paper is based on *shrinkage*, a notion presented by Dahlhaus *et al.* in [5] where an instance I is transformed into a smaller instance I' in such a way that all optimal solutions in I' are also optimal solutions in I.

In this paper, we generalize the shrinkage criterion of Dahlhaus et al. which is based on `st` cuts, to all nodes in the instance graph and prove its correctness. We also consider *maximum size* minimum `st` cuts, as this gives more efficient shrinkages, and we prove that the procedure of [8] for max-flow/min-cut actually computes these cuts. Then, we present an implementation of a fast local heuristics taking advantage of this new

* Work supported by the *Region de Bruxelles Capitale*, grant no. RBC-BR 227/3298.

shrinkage operation. The heuristics combines both the new shrinkage based reduction and an *unshackle* operation which operates on graphs where no more shrinkage is possible. A practical evaluation is presented which shows that our heuristics has generally better results than the approximation algorithm designed by Dahlhaus et al. To the best of our knowledge, our comparison is also the first experimental study of the approximation algorithm of [5].

2 Optimal static code distribution is hard

Our problem consists in finding, at compile time, an optimal distribution of an imperative regular program. Such a program contains instructions (assignments, loops and tests), a set of static global internal variables, a set of static global I/O variables, and a set of local mobile internal variables. This last set of variables is not relevant to the problem considered here and, from now on, we will only refer to static global variables when we talk about variables. The distributed environment in which the program runs is composed of several sites, each of which contains some of the global I/O variables. Each global I/O variable is assigned to exactly one site. A correct distribution is an assignment of all variables and all instructions to the set of sites such that the following distribution constraints are respected :

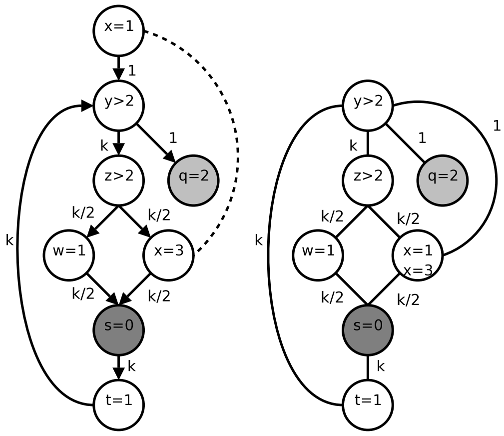
1. the I/O variables are on the predefined sites
2. each variable and instruction is on exactly one site
3. each instruction using a variable is on the site of that variable

Note that for some programs, a correct distribution does not exist because of the constraints on the I/O variables. From here on, we consider only correct programs (i.e. for which at least one distribution exists).

The assignment of instructions to sites influences the performance of the program during execution: each time control flows from an instruction assigned to one site to an instruction assigned to another site, the executed distribution environment must synchronize (e.g. by sending a message over a network) in order to continue execution on the other site. The optimal distribution is such that the expected number of (synchronization) messages created during execution is minimal. In order to evaluate this performance criterion, we suppose that a *realistic* control flow frequency function $\mathcal{W}$ is given, expressing the expected number of times control flows from one instruction to another. Although calculating such a function is undecidable in general, profiling tools and monitoring can result in good approximations. Note that a *realistic* control flow frequency function $\mathcal{W}$ for common languages will be such that $\mathcal{W}(i,i') = \mathcal{W}(i',i'')$ if i,i' and i'' are sequential instructions, $\mathcal{W}(i,i') = k\mathcal{W}(i'',i''')$ if i,i' is in a loop that is executed k times relative to the instructions i'',i''' outside the loop, and $\mathcal{W}(i,i') = \frac{1}{p}\mathcal{W}(i'',i'''), p \in [0,1]$ if i,i' is in a conditional `IF` branch with respect to instructions i'',i''' outside the `IF`. We suppose that $p \in \mathbb{Q}$ so that every weight can be multiplied by a common factor to obtain an equivalent instance of the problem with natural weights.

```
x=1;
while (y>2) { // k times
  if (z>2) w=1; // 1/2
  else x=3;     // 1/2
  s=0;
  t=1;
}
q=2;

I/O  q : site1;
I/O  s : site2;
```



a. **b.** **c.**

Fig. 1. The distribution problem

Definition 1 (The optimal distribution problem). *Given a certain program, a set of I/O variables, a set of sites and an assignment of the I/O variables to the sites, find a total assignment of all global variables and instructions to the set of sites, respecting the distribution constraints, such that the sum of the expected number of times control flows between instructions on different sites is minimal.*

A graphical presentation of the optimal distribution problem can be found in figure 1. The program of figure 1.a can be graphically modeled by its control flow graph (figure 1.b) where the nodes are its instructions and edges model the control flow between instructions with weights defined by $\mathcal{W}$. The graph of figure 1.c is the undirected graph where all nodes using the same variables are merged. We say that this graph is the result of the merging of `x=1` and `x=3` in the first graph. Remark that both representations are equivalent with respect to the optimal distribution problem. The merging operation is sometimes called *contraction* if an edge exists between two merged nodes, since that edge would disappear from the graph. We now give a formal definition of the multiterminal cut problem on weighted undirected graphs.

Definition 2 (Multiterminal cut problem). *Given a weighted undirected graph* $G(V, E, w) : E \subseteq \{\{u, v\} | u, v \in V \wedge u \neq v\}$ [1], $w : E \mapsto \mathbb{N}$ *and a set of terminals* $T = \{s_1, ..., s_k\} \subseteq V$, *find a partition of* V *into* $V_1, ..., V_k$ *such that* $s_i \in V_i \ \forall i \in [1, k]$ *and* $\sum_{v \in V_i, v' \in V_j, i \neq j} w(v, v')$ *is minimized.*

We know that the multiterminal cut problem is NP-Hard [5] for fixed $k > 2$, even when all weights are equal to 1. We now show that both problems are equivalent.

Theorem 1. *There exists a polynomial-time reduction from the optimal distribution problem to the multiterminal cut.*

Proof. Take the control flow graph G of the optimal distribution problem weighted by $\mathcal{W}$, and merge all instructions using the same variables into one node. This way, we only have at most one node using a given variable. Let G' be this new graph, and let $T = \{s_1, s_2, \ldots, s_k\}$ be the set of nodes of G' using a I/O variable. It is easy to see that G', T is an instance of the multiterminal cut and that an optimal solution to this instance is an optimal distribution of the original program. Indeed, starting from a multiterminal cut $V_1, \ldots, V_k$, we can build an optimal partition $P_1, \ldots, P_k$ such that P_i is the set of instructions such that their nodes in the control flow graph is in V_i. Note that G', T can be computed from G in polynomial time. ∎

The first theorem states that we can easily solve the optimal distribution problem using multiterminal cut. Next lemma states that, starting from an unweighted graph, we can build a program such that the control flow graph, after merging, is equal (except for a factor 2 w.r.t. the weights) to the original graph.

Corollary 1. *Let* P *be a program, and* G_f *its control flow graph after merging, it is equivalent to solve the optimal distribution on* P *or to solve the multiterminal cut on* G_f.

Proof. If we have an optimal distribution $P_1, \ldots, P_k$ of P, then we can easily compute a multiterminal cut $V_1, \ldots, V_k$ for G_f by taking the same partition. The constraint that every vertex of G_f is in only one S_i is respected because of constraint 2 of the definition of distribution. Moreover,

[1] For technical reasons looping edges (v, v) will be omitted in all graphs considered here. Note that their presence does not change the problem.

it is easy to see that $V_1, \ldots, V_k$ is optimal multiterminal cut. The other direction is a direct consequence of theorem 1.

Next lemma states that, starting from an unweighted graph, we can build a program such that the control flow graph, after merging, is almost equal to the original graph.

Lemma 1. *Let $G(V, E)$ be an unweighted graph, then there exists a program P such that the control flow graph $G_f = (V_f, E_f, w_f)$ of P, after merging, is such that $V = V_f$, $E = E_f$, and $\forall \{v, v'\} \in E,\ w_f(\{v, v'\}) = 2$.*

Proof. Let $G(V, E)$, we can build a program P such that, after having merged nodes using the same variables in the control flow graph, the resulting graph G_f is equal to G. Moreover, this program contains only a list of assignments. We define the set of global variables as $X_V = \{x_v |\ v \in V\}$, i.e. there is a bijective mapping between the set V and the set of global variables X_V. We show the construction of P by induction on the size of V. The case where $V = \{v\}$ (i.e. $|V| = 1$) is obvious, we only have one instruction $i \equiv x_v \leftarrow 1$.
Suppose therefore that $|V| = n$, and let P be the corresponding program using instructions $\{x_u \leftarrow 1 | u \in V\}$. We now build a program P' for the graph $G' = (V' = V \cup \{v\}, E' = E \cup \{\{v, v_1\}, \ldots \{v, v_\ell\}\}$. For this, we modify P as follows :
$\forall j \in [1, \ell]$ (i.e. for all new edges), let x_{v_j} be the global variable corresponding to v_j, and let $i \equiv x_{v_j} \leftarrow 1$ be an instruction of P. In P', we replace i by : $i; x_v \leftarrow 1; i$. Note that P' is still a list of assignments. Let G'_f be the control flow graph of P', after merging, we have that G'_f contains all edges of E'. And, if we set $\mathcal{W}$ such that it assigns 1 between two consecutive instructions (which is the case for common programming languages), then all edge weights in G'_f will be equal to 2. ■

Note that the number of assignments of P is polynomial in the size of G. Since all edges in G_f have weights equal to 2, it is easy to see that G and G_f has the same optimal solution for the multiterminal cut.

Theorem 2. *There exists a polynomial time reduction from multiterminal cut on unweighted graphs to the optimal distribution problem.*

Proof. Let $G(V, E), T$ be an unweighted instance of multiterminal cut, by lemma 1, we can build a program P which has its control flow graph G_f after merging equal to G (except for the constant factor of 2 on the weights). From corollary 1, we know that it is equivalent to solve the multiterminal cut on G_f than to solve the optimal distribution on P.

We know that multiterminal cut on unweighted graphs is NP-hard. Thus, we can state the following corollary.

Corollary 2. *The optimal distribution problem is hard, even when the program contains only a sequence of instructions for the multiterminal cut.*

Corollary 3. *The optimal distribution problem is equivalent to the multiterminal cut problem on arbitrary graphs.*

This results from theorems 1 and 2. Arbitrary weights 2ω can be obtained in the construction of theorem 2 using the `while` construct as follows : replace the sequence $i; x_v \leftarrow 1; i$ with $i; x_v \leftarrow w$; `while(` $x_v > 1$ `) {` $i; x_v \leftarrow x_v - 1$ `}` i.

3 Related Works

The multiterminal cut problem has first been studied by Dahlhaus *et al.* in [5]. In this paper, the authors prove that this problem is NP-hard for $k > 2$ even when k is fixed where k is the number of terminals. The problem is polynomially solvable when $k = 2$, a well known result proved by Ford and Fulkerson [6], and in the case of planar graphs. The authors also present a $2 - \frac{2}{k}$ polynomial time approximation algorithm that relies on *isolating* cuts, a technique that is detailed further on. Moreover, they proved that this problem is MAX SNP-hard, i.e. there is no polynomial time approximation scheme unless P=NP. In [1], Calinescu, Karloff, and Rabani, presented a linear programming relaxation. Using this technique and a well chosen rounding procedure, they obtain an approximation factor of $1.5 - \frac{1}{k}$. This factor was lowered to 1.3438 by Karger *et al.* in [10] who give better approximations when $k \geq 14$. These improvements were found by studying carefully the integrality gap and giving a more precise rounding procedure. A polyhedral approach [12, 13, 3] and a non-linear formulation [4] has also been studied for the multiterminal cut problem.

Shrinkage has also been studied by Högstedt and Kimelman in [9]. In this paper, the authors give some optimality-preserving heuristics that allow to reduce the size of the input graph by contracting some edges. The shrinkage technique presented here generalizes some of their criteria (such as independent nets and articulation points).

In this paper, we consider the multiterminal cut problem on undirected graphs, but work has also been done on directed graphs. Naor and Zosin presented a 2-approximation algorithm for this problem in [11]. On the other hand, Costa, Letocart and Roupin proved in [2] that multiterminal cuts on acyclic graphs could be computed in polynomial time using a simple flow algorithm. A generalization of multiterminal cut is *minimum multicut* where a list of pairs of terminals is given and we must find a set of edges such that these pair of terminals are disconnected. Garg *et al.* [7] give a $O(\log k)$-approximation algorithm for this minimum multicut. A survey on multiterminal cuts and its variations can be found in [2].

The applications that rely on the multiterminal cut fall mainly into two domains : the domain of parallel computation and the partitioning of distributed applications. The problems encountered in parallel computation are concerned with the allocation of loads on different processors. The total load must be partitioned in roughly equal sized pieces, characterized by some load balancing criterion, and this subject to some interconnection criterion that must be minimized. These problems can be formulated using the strongly related k-cut problem, which asks to partition the graph in k subsets such that crossing edges are minimized. Since this problem has no fixed terminals, it is polynomially solvable, for any fixed $k \geq 3$ and is therefore considerably easier than the problem addressed here.

For the distributed applications, the problem consists in partitioning the application in several components that must be distributed among different processors. Several criterion are studied, such as the inter object communication load of [9]. However, we are not aware of other works that are based on the static distribution of the instructions where the control flow is used to minimize the expected communications load. Because of this fine grain distribution, the scale of our problem is considerably larger than the studies on the partitioning of objects or functions as is the case in *classical* distributed systems. Therefore, we believe that the results of the heuristics presented here are applicable on these smaller instances as well.

4 A generalized global criterion

In [5] the authors design a $2 - \frac{2}{k}$ approximation algorithm based on the isolation heuristics which uses `st` cuts. An `st` cut (multiterminal cut with $k = 2$) divides the graph into two sets $(C, \overline{C})$ where $s \in C$ and $t \in \overline{C}$. The heuristics consists in finding an optimal isolating cut for each of the k terminals $\{s_1, ..., s_k\}$ and taking the union of the $k-1$ smallest of these cuts. An optimal isolating cut is a minimal `st` cut where s=s_i and t is the node resulting of the merging of $s_{j \neq i}$. We now introduce the original shrinkage theorem proved by Dahlhaus et al :

Theorem 3 (Shrinkage). *Given graph $G(V, E, w)$ with terminals $T = \{s_1, ..., s_k\} \subseteq V$. Let G'_i be the graph where all terminals in $T \setminus \{s_i\}$ are merged into t, and $(C, \overline{C})$ the* st *cut between s_i and t, then there exists an optimal multiterminal cut $(V_1, ..., V_k)$ of G such that $\exists \ell : C \subseteq V_\ell$.*

Theorem 3 allows us to shrink (i.e. to merge) all nodes in C into one node. Shrinkage is clearly an interesting way to attack the multiterminal cut problem. If one can obtain enough shrinkage to obtain a relatively small instance, then there may be hope to find the optimal solution by exhaustive search. It can also be used independently of any other algorithm designed to approximate the multiterminal cut problem. We extend theorem 3 to handle more shrinkage as follows :

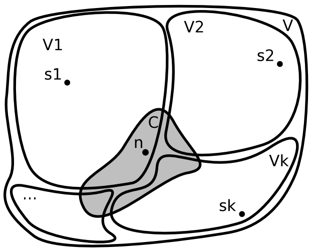


Fig. 2. Theorem 4

Theorem 4 (More shrinkage). *Given graph $G(V, E, w)$ with terminals $T = \{s_1, ..., s_k\} \subseteq V$. Let $v \in V$, and G'_i be the graph where all terminals in $T \setminus \{v\}$ are merged into t, and $(C, \overline{C})$ the minimum* st *cut between v and t in G'_i then there exists an optimal multiterminal cut of G $(V_1, ..., V_k)$ such that $\exists \ell : C \subseteq V_\ell$.*

Proof. Outline (a detailed proof can be found in appendix A). Figure 2 illustrates the proof for $l = 1$. Proof by contradiction. Take any minimal multiterminal cut $C^* = (V_1, V_2, ..., V_k)$ and suppose that $v \in V_1$ but $C \not\subseteq V_1$. Take $C^{*'} = (V_1 \cup C, V_2 \setminus C, ..., V_k \setminus C)$. Then we can show that the weight of $C^{*'} \leq$ the weight of C^* by using the weights of the edges between C and $\overline{C}$ and between $C \cap V_1$ and $\overline{C \cap V_1}$: since $(C, \overline{C})$ is a minimal st cut, the border of V_1 can be extended to $V_1 \cup C$ without increasing the multiterminal cut's weight. ■

To shrink an instance of the multiterminal cut with $|V| = n, |E| = m$, using theorem 4, one can use the algorithm of Goldberg and Tarjan [8] to calculate n st cuts each of which takes $O(nm \log \frac{n^2}{m})$, resulting in a total complexity of $O(n^2 m \log \frac{n^2}{m})$.

Once the graph cannot be reduced any further, two options remain, either search exhaustively, or *unshackle* the graph by contracting one or more edges that likely connect nodes from the same partition in the optimal cut. Once the graph is *unshackled*, the graph may be ready for further reductions. An implementation of the second solution, using a fast local unshackling heuristics is presented and evaluated in the following section.

5 A fast local heuristics

In this section we present a fast local unshackling heuristics for the multiterminal cut. First we present some observations on the `st` cut, that allow us to execute the unshackling heuristics of figure 3 more efficiently. Next, we show practical results.

```
reduce();
while(non-terminals exist) {
    unshackle(); // Contract 1 edge
    reduce();
}
```

Fig. 3. Unshackling heuristics

5.1 Definition and complexity

Definition 3 (MAX-MIN-`st` cut). *Given graph $G(V, E, w)$ and two different nodes $s, t \in V$. Define min-$ST(s, t)$ as the set of cuts separating s and t with minimal weight. We define the set* MAX-MIN-`st` (s, t) *as the set of nodes left connected to s by the cut $(C, \overline{C}) \in$ min-$ST(s, t)$ such that $|C|$ is maximal. We can easily extend these definitions for sets of nodes. For a set T,* MAX-MIN-`st` (s, T) *is equivalent to* MAX-MIN-`st` (s, t) *in the graph G where all nodes in T have been merged into the new node t.*

We prove the following properties: (Full proofs can be found in appendix A)

Theorem 5. *Given graph $G(V, E, w)$ and two nodes $s, t \in V$,* MAX-MIN-`st`(s, t) *is uniquely defined, i.e. there is only one maximal size minimum* `st` *cut for any couple (s, t)*

Theorem 6. *For any three nodes s, s', t of V, if $s' \in$* MAX-MIN-`st`(s, t)*, then* MAX-MIN-`st` $(s', t) \subseteq$ MAX-MIN-`st` (s, t).

Theorem 7. *Given graph $G(V, E, w)$ and three distinct nodes $s, s', t \in V$. Let $S = \text{MAX-MIN-}\mathtt{st}(s, t)$, $S' = \text{MAX-MIN-}\mathtt{st}(s', t)$, $I = S \cap S'$, and $T = V \setminus (S \cup S')$. If $I \neq \emptyset \wedge S \neq I \wedge S' \neq I$, then $w(I, S \setminus I) = w(I, S' \setminus I)$. Moreover, we have that $w(I, V \setminus (S \cup S')) = 0$. The same results holds when $S = \text{MAX-MIN-}\mathtt{st}(s, \{t \cup s'\})$, $S' = \text{MAX-MIN-}\mathtt{st}(s', t)$ or $S = \text{MAX-MIN-}\mathtt{st}(s, \{t \cup s'\})$, $S' = \text{MAX-MIN-}\mathtt{st}(s', \{s \cup t\})$.*

Theorems 5, 6 and 7 allow us to efficiently calculate the reduction phases of our unshackling heuristics. We know that the order in which we calculate the cuts has no effect on the outcome of the algorithm. Moreover, we can calculate the MAX-MIN-`st` cut for a given node n and immediately merge all nodes on the same side of n in the cut, thus reducing the number of nodes before calculating the next MAX-MIN-`st` cut for the remaining unmodified nodes. After the calculation and merging of all MAX-MIN-`st` cut, we have for all nodes in the reduced graph and terminals $s_1, \ldots, s_k$, MAX-MIN-`st` cut $(s, \cup_i s_i \setminus \{s\}) = \{s\}$. The only missing link is how to calculate MAX-MIN-`st`:

Theorem 8. *The algorithm of Goldberg and Tarjan [8] calculating the maximum flow in $O(nm \log(\frac{n^2}{m}))$-time also calculates* MAX-MIN-`st`

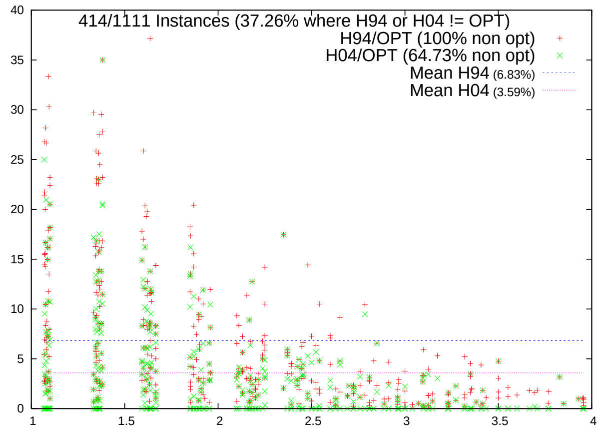
Finally, we can prove that the worst execution time for the unshackling heuristics stays within the complexity of the reduction algorithm :

Theorem 9. *The unshackling algorithm from figure 3 can be implemented with worst case complexity $O(n^2 m \log(\frac{n^2}{m}))$ if* `unshackle()` *has lower complexity than $O(nm \log(\frac{n^2}{m}))$.*

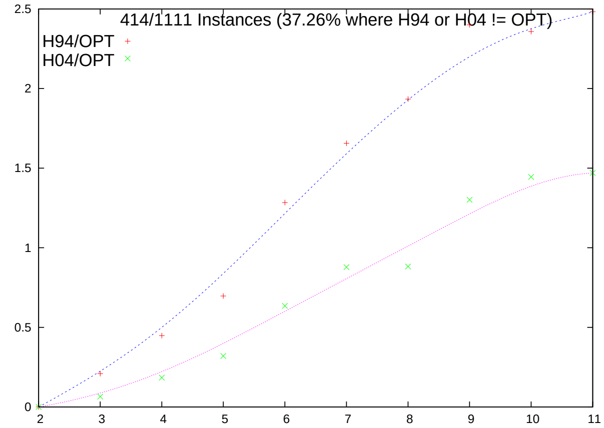
Proof. Consider an irreducible graph in which one and only one edge $\{v_1, v_2\}$ is contracted. Before contraction, $\forall v \in V$: MAX-MIN-`st` $(v) = \{v\}$. It is easy to see (proof by contradiction) that after contraction $\forall v \in V \setminus \{v_1, v_2\}$: MAX-MIN-`st` $(v) = \{v\}$, i.e. the contraction only affects the resulting node from the contraction, which means that after each contraction only one MAX-MIN-`st` cut has to be calculated. Since at most n contractions are possible, the number of MAX-MIN-`st` calculations needed can be bounded by $2n$ (n for the initial reduction and n for all subsequent reductions). The worst case complexity is therefore as stated, if `unshackle()` has lower complexity than $O(nm \log(\frac{n^2}{m}))$. ∎

5.2 Results

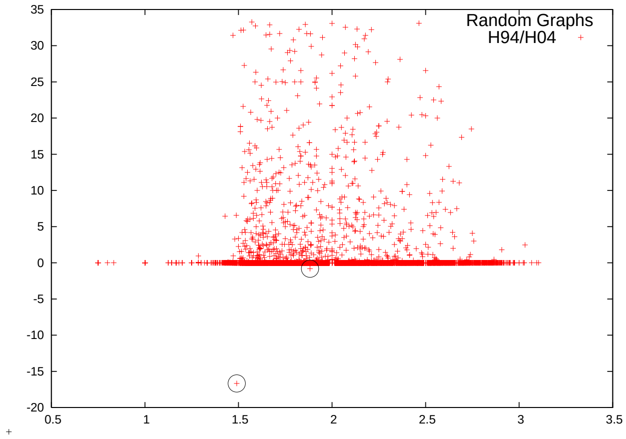
It remains to define the way we will unshackle the graph. We tried several local procedures, among which :



(a) Unshackling heuristics on random graphs



(b) Unshackling heuristics on random graphs



(c) Unshackling heuristics on grammar graphs

Fig. 4. Results for the unshackling heuristics

– *Greedy* : take an edge with maximal weight.
– *Error-reduction* : take an edge where the expected error is small
– *Balanced weight* : take an edge $\{n_1, n_2\}$ such that $\sum_{\{\{n_1,n'\}\in E\}} w(n_1, n') + \sum_{\{\{n_2,n'\}\in E\}} w(n_2, n')$ is maximal
– *Max-unshackle* : take an edge that has high reduction rate

Surprisingly, we found that *balanced weight* works much better than the others. Unfortunately, we discovered that none of these heuristics have a fixed approximation bound, however, since our calculations include the ones from the $k - \frac{2}{k}$ approximation algorithm, we can compare the results and take the better of both, resulting in the same bound without any extra cost. Note however that in order to compare both heuristics, this has not been done in the following experiments.

Two sets of experiments where conducted : figures 4(a), 4(b) show the results on random graphs, while 4(c) shows the results on graphs obtained from auto generated $_{\mathrm{d}}$SL programs.

In figure 4(a) we compare the results of our heuristics (indicated by $\mathcal{H}04$) with the approximation algorithm (called $\mathcal{H}94$) from Dahlhaus et al. 1111 experiments where conducted on sufficiently small graphs (ranging from 20-40 nodes). The small size allows us to compare the results with the optimal solution. For 411 *hard* cases (37%), one of the heuristics failed to find the optimal. We can see that for increasing mean degree (X-axis), the error rate (Y-axis, in percent w.r.t. the optimal) for both algorithms drops rapidly, caused by the randomness in the graph. For sparse graphs however, error rates can be as high as 35%. The mean error rate for $\mathcal{H}04$, for these *hard* cases, is 3.6% while it raises to 6.8% for $\mathcal{H}94$. Remark that the failure rate for $\mathcal{H}94$ is 100% of the hard cases, while our algorithm failed in 65% of these cases. In these experiments, there was no instance where $\mathcal{H}04$ performed worse compared to $\mathcal{H}94$.

Figure 4(b) shows the mean error rate (Y-axis, in % w.r.t. the optimal solution) for the complete set of experiments, with increasing number of terminals (X-axis). We can clearly see the gain of our algorithm.

Figure 4(c) shows results for 25.000 grammar graphs of moderate size (600 nodes, 3 to 10 terminals), where the two algorithms are compared to each other. X-axis gives the mean degree. We can observe a difference of as high as +35% (Y-axis) for some cases, meaning that our algorithm improves the other by the same amount. For only 2 instances, our algorithm performed worse (1.3% worse and 14% worse).

6 Conclusions

In this paper, we study the problem of optimal code distribution of an imperative regular program. During this study, we prove that this problem is polynomially equivalent to the multiterminal cut problem.

We presented a criterion that allows to perform shrinkage on a given graph such that optimal solutions for the multiterminal cut problem on the resulting graph are optimal solutions for the original graph. This criterion is a generalization of the criterion of Dahlhaus *et al.* presented in [5]. Using this shrinkage criterion, we designed a heuristics that, in practice, finds near optimal solutions for the multiterminal cut problem.

Future works We are currently searching other unshackling procedures that could be combined with our shrinkage technique. Moreover, we are trying to determine why the *balanced weight* unshackling performs well in practice.

We are also considering the use of our shrinkage technique in the branch and bound context. We are currently integrating our shrinkage technique into the CPLEX solver in order to speed up the general branch and bound phase of the mixed integer optimizer.

References

1. Gruia Calinescu, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for multiway cut. *J. Comput. Syst. Sci.*, 60(3):564–574, 2000.
2. M. Costa, L. Letocart, and F. Roupin. Minimal multicut and maximal integer multiflow: a survey. *European Journal of Operational Research*, 162(1):55–69, 2005.
3. W.H. Cunningham. The optimal multiterminal cut problem. *DIMACS series in discrete mathematics and theoretical computer science*, 5:105–120, 1991.
4. R. Vohra D. Bertsimas, C. Teo. Nonlinear formulations and improved randomized approximation algorithms for multicut problems. In Springer-Verlag, editor, *Proc. 4th conference on integer programming and combinatorial optimization*, volume 920 of *LNCS*, pages 29–39, 1995.
5. Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, P. D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.
6. L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.
7. Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming*, pages 487–498. Springer-Verlag, 1994.
8. Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, October 1988. ISSN:0004-5411.
9. K. Hogstedt and D. Kimelman. Graph cutting algorithms for distributed applications partitioning. *SIGMETRICS Performance Evaluation Review*, 28(4):27–29, 2001.
10. David R. Karger. Random sampling in cut, flow, and network design problems. In *Proc. 26 th Annual ACM Symposium on the Theory of Computing*, pages 648–657, 1994.
11. J. Naor and L. Zosin. A 2-approximation algorithm for the directed multiway cut problem. *SIAM J. Comput.*, 31(2):477–482, 2001.

12. L.H. Owen S. Chopra. Extended formulations for the a-cut problem. 1994.
13. M.R. Rao S. Chopra. On the multiway cut polyhedron. *Networks*, 21:51–89, 1991.
14. Bram De Wachter, Thierry Massart, and Cédric Meuter. dsl : An environment with automatic code distribution for industrial control systems. In M. Papatriantafilou and Ph. Hunel, editors, *Principles of Distributed Systems, 7th International Conference, OPODIS 2003*, volume 3144 of *LNCS*, pages 132–145. Springer, 2004.

A Proofs

Theorem 4 (More shrinkage). *Given graph $G(V, E, w)$ with terminals $T = \{s_1, ..., s_k\} \subseteq V$. Let $n \in V$, and G'_i be the graph where all terminals in $T \setminus \{n\}$ are merged into t, and $(C, \overline{C})$ the min-ST-Cut between n and t then there exists an optimal multiterminal cut of G $(V_1, ..., V_k)$ such that $\exists \ell : C \subseteq V_\ell$.*

Proof. Suppose a minimal multiterminal cut C^* and, without any loss of generality, $l = 1(i.e. v \in V_1)$. We define a partition $C^{*'}$ of V in $V'_1, ..., V'_k$ as follows :

$$V'_1 = V_1 \cup C \tag{1}$$

$$V'_{j\neq 1} = V_j \setminus (V_j \cap C) = V_j \setminus C \tag{2}$$

We show that $C^{*'}$ is a multiterminal cut with weight less than (or equal to) the weight of C^*, proving the existence of a multiterminal cut described in the theorem. It is easy to verify that $C^{*'}$ is a multiterminal cut , since

- $V'_i \cap V'_{j\neq i} = \emptyset$
- $\cup_i V'_i = V$
- $\forall i : s_i \in V'_i$
- $\forall i : s_i$ is isolated from $s_{j\neq i}$

Let us show that $C^{*'}$ has weight less than (or equal to) the weight of C^*. Let $A, B \subseteq V,\ A \cap B = \emptyset$, we note $w(A, B) = \sum_{\{(x,y)|x\in A, y\in B\}} w(x, y)$. Furthermore, let $w(X) = w(X, \overline{X})$, where $\overline{X} = V \setminus X$. Using these notations, we have the following rules :

$$w(X, Y) = w(Y, X) \tag{3}$$

$$w(X, Y \cup Z) = w(X, Y) + w(X, Z) - w(x, Y \cap Z) \tag{4}$$

$$w(X, Y \cup Z) = w(X, Y) + w(X, Z) \qquad \text{if } Y \cap Z = \emptyset \tag{5}$$

$$w(X, Y \setminus Z) = w(X, Y) - w(X, Y \cap Z) \tag{6}$$

The weights of the two multiterminal cuts $C^{*'}$ and C^* can be expressed in terms of their partitions (resp. V'_i and V_i) as follows :

$$w(C^{*'}) = \sum_{j\neq 1} w(V'_1, V'_j) + \sum_{i\neq 1, j>i} w(V'_i, V'_j) \tag{7}$$

$$w(C^*) = \sum_{j\neq 1} w(V_1, V_j) + \sum_{i\neq 1, j>i} w(V_i, V_j) \tag{8}$$

In addition, we use two `st` cuts for proving $w(C^{*'}) \leq w(C^*)$:

$$
\begin{aligned}
w(C) &= w(C, V \setminus C) = w(C, (\cup_j V_j) \setminus C) && (9)\\
&= \sum_j w(C, V_j \setminus C) && \text{by (4), (5)}\\
&= w(C, V_1 \setminus C) + \sum_{j \neq 1} w(C, V_j \setminus C) && (10)\\
w(C \cap V_1) &= w(C \cap V_1, V \setminus (C \cap V_1)) && (11)\\
&= w(C \cap V_1, (\cup_j V_j) \setminus (C \cap V_1)) && (12)\\
&= \sum_j w(C \cap V_1, V_j \setminus (C \cap V_1)) && \text{by (4), (5)}\\
&= w(C \cap V_1, V_1 \setminus C) + \sum_{j \neq 1} w(C \cap V_1, V_j) && (13)
\end{aligned}
$$

In order to calculate $w(C^{*'}) - w(C^*)$, we will express all terms of $C^{*'}$ in terms of C^* :

$$
\begin{aligned}
w(V_1', V_{j\neq 1}') &= w(V_1 \cup C, V_j \setminus C) && \text{by (1), (2)}\\
&= w(V_j \setminus C, V_1) + w(V_j \setminus C, C) \\
&\quad - w(V_j \setminus C, V_1 \cap C) && \text{by (3), (4)}\\
&= w(V_1, V_j) - w(V_1, V_j \cap C) + w(V_j \setminus C, C) \\
&\quad - w(V_1 \cap C, V_j) + w(V_1 \cap C, V_j \cap C) && \text{by (3), (6)}\\
& && (14)\\
w(V_{i\neq 1}', V_{j>i}') &= w(V_i \setminus C, V_j \setminus C) && \text{by (2)}\\
&= w(V_i \setminus C, V_j) - w(V_i \setminus C, V_j \cap C) && \text{by (6)}\\
&= w(V_j, V_i) - w(V_j, V_i \cap C) \\
&\quad - w(V_j \cap C, V_i) + w(V_j \cap C, V_i \cap C) && \text{by (3), (6)}\\
& && (15)
\end{aligned}
$$

Using equations (14) and (15) in (7), we can express the difference between $C^{*'}$ and C^* using (10) and (13) :

$w(C^{*'}) - w(C^*) =$

$$
\begin{aligned}
& w(C) - w(C \cap V_1) && (16)\\
& + \sum_{j \neq 1} (-w(V_1, V_j \cap C) + w(V_1 \cap C, V_j \cap C)) && (17)\\
& + \sum_{i \neq 1, j > i} (-w(V_j, V_i \cap C) - w(V_j \cap C, V_i) + w(V_i \cap C, V_j \cap C)) && (18)\\
& - w(V_1 \setminus C, C) + w(V_1 \cap C, V_1 \setminus C) && (19)
\end{aligned}
$$

which proves the theorem since

- (16) ≤ 0 because C is a minimal cut (remark that $C \cap V_1$ is a `st` cut between v and t)
- (17) ≤ 0 because $(V_1 \cap C) \subseteq V_1$
- (18) ≤ 0 because $(V_i \cap C) \subseteq V_i$ and $w(X, Y) \geq 0$
- (19) ≤ 0 because $(V_1 \cap C) \subseteq C$ ■

Theorem 5. *Given graph $G(V, E, w)$ and two nodes $s, t \in V$,* MAX-MIN-`st`(s, t) *is uniquely defined.*

Proof. By contradiction. Suppose S and $S'(S \neq S')$ both satisfy the definition of MAX-MIN-`st`(s, t) Let $I = S \cap S'$ and $T = V \setminus (S \cup S')$, it is easy to see that S (resp. S') $\neq I$, else S (resp. S') would not be a maximal size minimum `st` cut. As S is a min `st` cut, we have :

$$
\begin{aligned}
& w(I) \geq w(S)\\
\iff & w(S \setminus I, I) + w(S' \setminus I, I) + w(I, T) \geq w(S, S' \setminus I) + w(S, T)\\
\iff & w(S \setminus I, I) + w(S' \setminus I, I) + w(I, T) \geq w(S, S' \setminus I) + w(S \setminus I, T) + w(I, T)\\
\iff & w(S \setminus I, I) + w(S' \setminus I, I) \geq w(S, S' \setminus I) + w(S \setminus I, T)
\end{aligned}
$$

As $I \subseteq S$, we have that

$$w(S' \setminus I, I) \quad \big(= w(I, S' \setminus I)\big) \ \leq w(S, S' \setminus I)$$

Therefore, we must have

$$w(S \setminus I, I) \geq w(S \setminus I, T) \qquad (20)$$

Now, let's compute $w(S \cup S')$, note that $S \cup S'$ is also a $\texttt{st}$ cut for (s, t).

$$\begin{aligned}
w(S \cup S') &= w(S \setminus I, T) + w(S' \setminus I, T) + w(I, T) \\
&\leq w(S \setminus I, I) + \underbrace{w(S' \setminus I, T) + w(I, T)}_{= w(S', T)} && \text{by 20} \\
&\leq w(S \setminus I, I) + w(S', T) \\
&\leq w(S \setminus I, S') + w(S', T) && \text{since } I \subseteq S' \\
&\leq w(S')
\end{aligned}$$

Thus, $S \cup S'$ is a minimal $\texttt{st}$ cut for (s, t). But we have $S' \subsetneq S \cup S'$, therefore $S, S' \notin$ MAX-MIN-$\texttt{st}(s, t)$ and we have a contradiction. ∎

Theorem 6. *For any three nodes s, s', t of V, if $s' \in$ MAX-MIN-$\texttt{st}(s, t)$, then max-min-ST$(s', t) \subseteq$ max-min-ST(s, t).*

Proof. By contradiction. Let $S =$ MAX-MIN-$\texttt{st}(s, t)$, $S' =$ MAX-MIN-$\texttt{st}(s', t)$ and suppose that $S' \not\subseteq S$. We have that $|S \cup S'| > |S|$. Let $I = S \cap S'$ and $T = V \setminus (S \cup S')$, we define the following :

$$A \equiv w(S \setminus I, T) \tag{21}$$
$$B \equiv w(I, T) \tag{22}$$
$$C \equiv w(S' \setminus I, T) \tag{23}$$
$$D \equiv w(I, S \setminus I) \tag{24}$$
$$E \equiv w(I, S' \setminus I) \tag{25}$$
$$F \equiv w(S \setminus I, S' \setminus I) \tag{26}$$

Figure 5 depicts these different sets of edges on an example. We have :

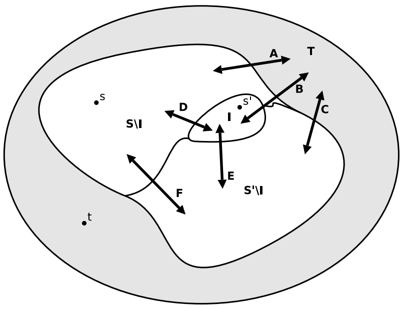


Fig. 5. The set of edges from equations 21 to 26

$$w(S \cup S') = A + B + C \tag{27}$$
$$w(S) = A + B + E + F \tag{28}$$
$$w(S') = B + C + D + F \tag{29}$$
$$w(I) = B + D + E \tag{30}$$

As $S = \text{MAX-MIN-}\texttt{st}(s,t)$ and $S' = \text{MAX-MIN-}\texttt{st}(s',t)$, we have the following :

$$w(S \cup S') > w(S) \qquad \text{As } S \subsetneq S \cup S' \tag{31}$$
$$w(I) \geq w(S') \tag{32}$$

which implies that :

$$C > E + F \qquad \text{By 31, 27 and 28} \tag{33}$$
$$E \geq C + F \qquad \text{By 32, 30 and 29} \tag{34}$$

Which leads to a contradiction. ■

Theorem 7. *Given graph $G(V,E,w)$ and three different nodes $s, s', t \in V$. Let $S = \text{MAX-MIN-}\texttt{st}(s,t)$, $S' = \text{MAX-MIN-}\texttt{st}(s',t)$, $I = S \cap S'$ and $T = V \setminus (S \cup S')$. If $I \neq \emptyset \wedge S \neq I \wedge S' \neq I$, then $w(I, S \setminus I) = w(I, S' \setminus I)$ and $w(I,T) = 0$. The same results holds when $S = \text{MAX-MIN-}\texttt{st}(s, \{t \cup s'\})$, $S' = \text{MAX-MIN-}\texttt{st}(s',t)$ or $S = \text{MAX-MIN-}\texttt{st}(s, \{t \cup s'\})$, $S' = \text{MAX-MIN-}\texttt{st}(s', \{s \cup t\})$.*

Proof. Proof by contradiction. Let's reuse equations 21 to 26 from proof of theorem 6. We now compute $w(S \setminus I)$ and $w(S)$:

$$w(S \setminus I) = A + D + F$$
$$w(S) = A + B + E + F$$

As S is the $\text{MAX-MIN-}\texttt{st}$ cut(s,t), we have :

$$A + B + E + F \leq A + D + F$$
$$B + E \leq D \tag{35}$$
$$\tag{36}$$

By applying a similar reasoning with S' and $S' \setminus I$, we can prove that $B + D \leq E$. In conclusion, we have $E = D (\Rightarrow w(I, S \setminus I) = w(I, S' \setminus I))$ and $B (\equiv w(I,T)) = 0$.
The two other propositions are proved by the same way. ■

Theorem 8. *The algorithm presented in [8] calculating the maximum flow in $O(nm\log(\frac{n^2}{m}))$-time also calculates* MAX-MIN-`st`

Proof. By contradiction
In [8] the authors prove that it is possible to calculate a minimal `st` cut $(S_g, \overline{S}_g)$ with $s \in S_g \wedge t \in \overline{S}_g$ in $O(nm\log(\frac{n^2}{m}))$-time. We will use their notations to prove that the min `st` cut calculated by their algorithm is in fact the unique minimal `st` cut of maximal size.
Let $g(v,w) : E \mapsto \mathbb{R}^+$ be the preflow function (here we may suppose that the algorithm terminated and that the preflow is a legal flow). G_g is used to indicate the residual graph and $c(v,w) : E \mapsto \mathbb{R}^+$ indicates the capacities of the edges in E. In addition, $(S_g, \overline{S}_g)$ is defined as the partition of V such that $\overline{S}_g$ contains all nodes from which t is reachable in G_g and $S_g = V \setminus \overline{S}_g$.
We use the following lemma from [8]:

> *When the first stage terminates, $(S_g, \overline{S}_g)$ is a cut such that every pair v, w with $v \in S_g$ and $w \in \overline{S}_g$ satisfies $g(v,w) = c(v,w)$.*

Suppose that there exists another cut $(C', \overline{C'})$ such that $|C'| > |S_g|$.
Remark that $S_g \subseteq C'$ because of theorem 6 and $s \in C' \cap S_g$.
Let $I = C' \cap \overline{S}_g$. Note that $I \neq \emptyset$ since $|C'| > |S_g|$. We split the boundaries between S_g, I and $\overline{S}_g$ in three sets : O (old boundary) , N (new boundary), and C (common boundary).

- $O \subseteq E = (v,w) : v \in S_g \setminus I \wedge w \in I$
- $N \subseteq E = (v,w) : v \in I \wedge w \in \overline{S}_g \setminus I$
- $C \subseteq E = (v,w) : v \in S_g \setminus I \wedge w \in \overline{S}_g \setminus I$

By definition of $(S_g, \overline{S}_g)$, we know that $g(O) = c(O)$. We also know that since $(C', \overline{C'})$ and $(S_g, \overline{S}_g)$ are both minimal cuts : $w(O)+w(C) = w(N)+w(C) \Rightarrow w(O) = w(N)$. Remark that since g is a legal flow, the flow entering I must be equal to the flow getting out of I, which means that $g(O) = g(N)$.

The combination of these tree equations leads to a contradiction: since the edges in N are saturated, t is not reachable from any $n \in I$ in G_g which means that $I = \emptyset$. ∎