A Fast Exact Algorithm for the Optimum Cooperation Problem [⋆]

Diana Fanghänel and Frauke Liers

Universität zu Köln, Institut für Informatik, Pohligstrasse 1, 50969 Köln, Germany

Abstract. Given a graph $G = (V, E)$ with edge weights $w_e \in \mathbb{R}$, the optimum cooperation problem consists in determining a partition of the graph that maximizes the sum of weights of the edges having nodes in the same partition plus the number of resulting partitions. The problem is also known in the literature as the optimum attack problem in networks. It occurs as a subproblem in the separation of partition inequalities. Furthermore, a relevant physics application exists.

Solution algorithms known in the literature require at least $|V| - 1$ minimum cut computations in a corresponding network. In this work, we present a fast exact algorithm for the optimum cooperation problem. By graph-theoretic considerations and appropriately designed heuristics, we considerably reduce the number of minimum cut computations that are necessary in practice. We show the effectiveness of our method by comparing the performance of our algorithm with that of the fastest previously known method on instances coming from the physics application.

Key words: optimum attack problem, submodular function minimization, minimum cut problem, partition inequalities, Potts glass with many states

1 Introduction

In this work, we will deal with the following optimization problem. Suppose the benefit of cooperation between two people, say, researchers, is represented by some number. Furthermore, there is a unit gain for each group working on, say, a research project. We search for an optimal cooperation, i.e., want to decide which researchers should collaborate in order to maximize the total benefit, see [1].

In graph-theoretic terms, the problem can be stated as follows. Let a graph $G = (V, E)$ with edge weights $w_e \in \mathbb{R}$ for the edges $e \in E$ be given. Vertices represent the researchers, edge weights correspond to the benefit of cooperation. We want to solve the problem

$$\max\{c_G(A) + w(A) : \ A \subseteq E\}, \tag{1.1}$$

where $w(A) = \sum_{e \in A} w_e$ and $c_G(A)$ is the number of connected components of the induced graph $G(A) = (V, A)$.

The problem was first studied by Cunningham in the context of determining optimum attacks in networks [5]. In this application, the weight w_e can be interpreted as a measure for the effort required by an attacker to destroy edge e. The task is to minimize the difference between the effort of destroying a set of edges and the number of newly generated components of the graph.

[⋆] Supported by the German Science Foundation under contract Li 1675/1. Partly supported by the Marie Curie RTN Adonet 504438 funded by the EU.

Another relevant application is the separation of partition inequalities, as introduced by Baïou, Barahona and Mahjoub [2]. Given a partition $\{S_1, \dots, S_p\}$ of the node set V, we denote by $\delta(S_1, \dots, S_p)$ the set of all edges having endnodes in different sets of the partition. Then, for given real numbers a and b, the inequality $w(\delta(S_1, \dots, S_p)) \leq ap + b$ is called partition inequality. The latter arise as valid inequalities for a number of combinatorial problems. In order to use them inside a cutting plane algorithm, we need to solve the separation problem that, given edge weigths $w_e \geq 0$, returns a partition violating the inequality, if it exists. Baïou et al. show that the separation problem can be solved by computing an optimal solution of (1.1).

An important model in statistical physics is the so-called Potts model [12]. It has been introduced as a generalization of the so-called Ising model to describe several physical systems. It is a model on a graph where the vertices represent magnetic spins. They are assigned variables that each can take values between $\{1, \dots, q\}$. Interactions between pairs of spins may be present. The aim is to compute the so-called partition function that encodes the physics of the system. For many relevant physics systems, computing the partition function is a difficult task. However, as pointed out by Juhasz, Rieger, Iglói [13] and Anglès d'Auriac, Iglói, Preissmann and Sebö [1], for big numbers q, determining the dominant contribution in the Potts partition function amounts to solving a problem of type (1.1).

Several solution algorithms for determining optimum cooperations or optimum attacks have been presented in the literature. As it is not hard to see that the function to be maximized is supermodular, any algorithm for submodular function minimization could be used to solve the problem. By now several polynomial algorithms [4, 8] are known to solve this task. However, the specific properties of the problem allow the usage of algorithms with better worst-case asymptotic running time.

Cunningham [5] developed the first combinatorial algorithm for the optimum attack problem that is based on $|E|$ minimum cut computations in an associated network. Thereafter, the worst-case running time of the algorithm was decreased to $|V|-1$ minimum cut computations by Baïou, Barahona and Mahjoub, and Barahona [2, 3]. Anglès d'Auriac et al. [1] built upon the existing work and presented an algorithm that also needs $|V| - 1$ minimum cut computations but is easier to implement. They presented some experimental results for instances coming from the physics application.

In this article we present an algorithm that is based on the one of Anglès d'Auriac et al. but has a better average running time, as by graph-theoretic considerations only a fraction of the minimum cut computations are necessary in practice. In Section 2 we define the necessary concepts. We also provide optimality conditions and theoretical results that prove the correctness of our algorithm. We give the proofs of the lemmas in the Appendix. In Section 3 we explain the algorithm of Anglès d'Auriac et al. Thereafter, we propose a modification yielding better performance. In Section 4, we explain details of the implementation and present experimental results on instances coming from the physics applications. Furthermore, we compare our algorithm with the original method of Anglès d'Auriac et al. It turns out that for many instances the running times can be reduced considerably.

2 Definitions and Theoretical Concepts

Let a graph $G = (V, E)$ with edge weights $w_e \in \mathbb{R}$ for all edges $e \in E(G)$ be given. Then

$$\max \left\{ f_{G,w}(A) = c_G(A) + \sum_{e \in A} w_e \;:\; A \subseteq E(G) \right\} \tag{2.2}$$

is called Potts problem and $f_{G,w}$ Potts function.

Obviously, deleting edges from G with nonpositive weights yields an equivalent problem. Analoguously, for an edge $e \in E(G)$ with $w_e \geq 1$, there always exists an optimal solution containing e. Thus, we obtain an equivalent problem by contracting all edges with weight at least 1. Therefore, w.l.o.g., we assume $w_e \in (0, 1)$ for all edges $e \in E(G)$.

It is not hard to see that the Potts function is supermodular [1]. A function f is called supermodular if for all subsets $A_1, A_2 \subseteq E$ the inequality $f(A_1 \cup A_2) + f(A_1 \cap A_2) \geq f(A_1) + f(A_2)$ is valid. If $(-f)$ is supermodular, f is called submodular. Submodular function minimization occurs in a huge number of different applications, see, e.g., [4]. Whereas it was already known for some time that the problem itself can be solved in polynomial time [9, 10], only recently, strongly polynomial combinatorial algorithms could be designed [4, 8, 7, 17]. Therefore, we could use any algorithm for minimizing submodular functions for solving (2.2). However, exploiting the structure of the Potts function yields a solution algorithm with better asymptotic running time. In order to do this, we need to investigate the Potts problem more intensively.
Let A^* be an optimal solution of the latter. It is easy to see that each connected component of $G(A^*)$ contains all edges of G it induces. In fact, let $X_1, \ldots, X_k \subseteq V(G)$ be the vertex sets of the connected components of the induced graph $G(A^*) = (V, A^*)$. If there is an edge $e = (u, v) \in E(G) \backslash A^*$ with both u, v contained in the same set X_i, adding e to A^* increases the value of the Potts function, in contradiction to the optimality of A^*. Therefore, it is possible to consider node partitions instead of edge sets.

Definition 2.1. *Let $\mathfrak{P}_G$ denote the set of all partitions of the node set V. Then a partition is called optimal if it solves the problem*

$$\max \left\{ F_{G,w}(\mathcal{P}) = |\mathcal{P}| + w(\mathcal{P}) \;:\; \mathcal{P} \in \mathfrak{P}_G \right\}, \tag{2.3}$$

where $|\mathcal{P}|$ denotes the number of classes and $w(\mathcal{P})$ the weight of all edges with endnodes in the same class of $\mathcal{P}$.

Obviously, if for a class X of a partition $\mathcal{P}$ the induced subgraph $G(X)$ is not connected, then the partition $\mathcal{P}$ is not optimal. Thus, there is a one-to-one correspondence between the optimal solutions of the Potts problem and the optimal partitions, i.e., the problems (2.2) and (2.3) are equivalent.

In the following, we will either speak of edge sets or of the corresponding partition, whatever is more natural in the context. In order to be able to design an efficient solution algorithm, we explain in the next section under which conditions the problem can either be decomposed or shrunk into smaller problems of the same type.

2.1 Decomposing the Problem and Shrinking Subgraphs

Because of the submodularity of the Potts function, there exists an optimum solution on G containing edge sets which are optimum for its induced subgraphs. This is stated in the next lemma which is a generalization from a lemma given in [1].

Lemma 2.1. *Let be given a graph $G = (V, E)$ and a vertex set $U \subset V(G)$ with $\emptyset \neq U$. Suppose further that $A_1 \subseteq E(G_1)$ resp. $A_2 \subseteq E(G_2)$ are optimal solutions of (2.2) for the induced subgraphs $G_1 = G(U)$ and $G_2 = G(V \backslash U)$, respectively. Then there exists an optimal solution $A^* \subseteq E(G)$ of the Potts problem for G with $A^* \supseteq A_1 \cup A_2$.*

Thus, we can obtain an optimal solution for G by a divide-and-conquer approach in which we first solve the problem for the smaller graphs $G(U)$ and $G(V \backslash U)$ and add further edges to the union of the optima for the smaller graphs. A special case is the choice of node sets having cardinality 1 for which Lemma 2.1 was proven in [1].

In order to formulate Lemma 2.1 using node partitions, let $X_1, \ldots, X_k$ be the vertex sets of the connected components of $G(A_1)$ and $Y_1, \ldots, Y_l$ the vertex sets of the connected components of $G(A_2)$. Then $\mathcal{P} = \{X_1, \ldots, X_k, Y_1, \ldots, Y_l\}$ is a partition of $V(G)$. Lemma 2.1 says that in an optimal solution A^* for G the vertex sets of the components of $G(A^*)$ are either classes or the union of classes of $\mathcal{P}$.

Knowing this decomposition lemma, we are interested in possible choices of the sets U. It turns out that sets defining cuts of small weight are important.

Lemma 2.2. *Let a set $U \subseteq V(G)$ with cut $\delta(U) := \{(v_1, v_2) \in E(G) : \ v_1 \in U, \ v_2 \notin U\}$ be given. Assume that the weight $w(\delta(U))$ of the cut is $w(\delta(U)) \leq 1$. Then there exists an optimal solution A^* of the Potts problem (2.2) with $\delta(U) \cap A^* = \emptyset$.*

Therefore, whenever we find a partition of the nodes of G into sets U and $V \setminus U$ with cut weight at most 1, we can decompose the problem into two problems of the same type, one defined on $G(U)$ and the other on $G(V \backslash U)$, that can be solved independently. An optimal solution for G then consists of the union of the edge sets that are optimum for $G(U)$ and $G(V \backslash U)$.

Apart from decomposing the problem, it is also possible to contract certain subgraphs of G. As a preprocessing step, we already proposed to contract edges $e \in E(G)$ with weight at least 1 before starting a solution algorithm. We now will generalize this procedure to the contraction of subgraphs of G.

In our context, contracting a subgraph $G(U)$ means replacing U by a supernode v_U. For all nodes $v \in V \backslash U$, we replace the set of all edges (v, u) with $u \in U$ by a single edge $e = (v, v_U)$ with weight $w_e := \sum_{(v,u):u \in U} w_{(v,u)}$. Edges (u_1, u_2) with $u_1, u_2 \in U$ are deleted.

Contracting an edge set $E' \subseteq E(G)$ means contracting the set of all incident vertices of these edges to a supernode and dealing with loops and multiple edges as before. G/U resp. G/E' denote the graphs generated by contracting the vertex set $U \subseteq V(G)$ resp. the edge set $E' \subseteq E(G)$ in G. In the next lemma we state the condition under which a subgraph can be contracted.

Lemma 2.3. *Let a partition $\mathcal{P} = \{P_1, \ldots, P_k\}$ of $V(G)$ be given. Furthermore, let $U \subseteq P_k$ be chosen such that $\tilde{\mathcal{P}} = \{U\}$ is optimal for $G(U)$. Then $\mathcal{P}$ is an optimal partition for G if and only if $\mathcal{P}' = \{P_1, \ldots, P_{k-1}, P_k/U\}$ is optimal for G/U.*

Lemma 2.3 can be used for the computation of an optimal partition as follows: If we find a subset $U \subseteq V(G)$ such that $\{U\}$ is optimum on $G(U)$, we proceed by solving the problem on G/U instead of G. G/U has a smaller number of vertices than G. Furthermore, some edge weights are increased by the contraction. As soon as they exceed the value 1, they can be contracted. In the next lemma, we give contraction criteria for some special graph structures.

Lemma 2.4. *1. Let a cycle C^n with $n \geq 3$ vertices and edge weights $w_e \in (0,1)$ for all $e \in E(C^n)$ be given. If $\sum_{e \in E(C^n)} w_e \geq n-1$, the edge set $E(C^n)$ is an optimal solution of (2.2).*

2. Let F_2 denote a cycle C^4 consisting of four edges with a chord e_0 inducing two triangles with edge sets $\{e_0, e_1, e_2\}$ and $\{e_0, e_3, e_4\}$, resp. Let $w_e \in (0,1)$ for all $e \in E(F_2)$. Then, $\{V(F_2)\}$ is an optimal partition for F_2 if the weights satisfy $w_{e_0} + w_{e_1} + w_{e_2} + w_{e_3} + w_{e_4} \geq 3$, $w_{e_1} + w_{e_2} \geq 1$ and $w_{e_3} + w_{e_4} \geq 1$.

3. $\{V(K^n)\}$ is an optimal partition for the complete graph K^n with n nodes if the weights satisfy $w_e \geq 2/n$ for all edges $e \in E(K^n)$.

In general, the subgraphs we find in a graph are often not induced ones. In the following lemma we show that this fact does not cause problems.

Lemma 2.5. *Let $U \subseteq V(G)$ be given and $\{U\}$ an optimal partition for $G(U)$ with edges weights w'_e for all edges $e \in E(G)$. Then $\{U\}$ is also optimal for $G(U)$ with edges weights w_e chosen as $w_e \geq w'_e$ for all edges $e \in E(G)$.*

This lemma can be used as follows. Assume G contains a subgraph G' with $V = V(G')$ and $\{V(G')\}$ is optimal for G'. Then the consideration of G' equals that of G, where the weights are chosen as $w'_e := w_e$ for all $e \in E(G')$ and $w'_e := 0$ for the edges $e \in E(G) \backslash E(G')$. Then Lemma 2.5 states that $\{V(G)\}$ is an optimal partition of $V(G)$.

Given these statements, a subgraph satisfying one of the above-mentioned conditions can be contracted using Lemma 2.3.

2.2 Optimality conditions

In this subsection we investigate optimality conditions for the Potts problem. We closely follow the considerations in [1, 12] and generalize some results from [1].

Let $U \subseteq V(G)$ be a vertex set such that $\mathcal{P}' = \{U\}$ is an optimal partition for $G(U)$, and let $\mathcal{P}'' = \{X_1, \dots, X_k\}$ be optimal for $G(V \backslash U)$. Then $\mathcal{P} = \{X_1, \dots, X_k, U\}$ is a partition for the graph G. It turns out that we can obtain an optimal partition for G from the latter by merging the set U with classes of $\mathcal{P}$. In the next lemma we state how such a union of classes changes the value of a partition.

Lemma 2.6 ([1]). *Let $\mathcal{P}$ and $\mathcal{P}^*$ be two partitions of G with $\mathcal{P}^* = (\mathcal{P} \backslash \mathcal{W}) \cup \{\{\bigcup X_i : X_i \in \mathcal{W}\}\}$ for a set $\mathcal{W} \subseteq \mathcal{P}$. Then $f_{G,w}(\mathcal{P}^*) = f_{G,w}(\mathcal{P}) - |\mathcal{W}| + 1 + w(E(\mathcal{W}))$, where $E(\mathcal{W})$ is the set of all edges between the vertex sets $X_i, X_j \in \mathcal{W}$, $i \neq j$.*

It follows that a partition $\mathcal{P}$ is optimal if and only if the inequality $|\mathcal{W}| - 1 - w(E(\mathcal{W})) \geq 0$ is valid for all $\mathcal{W} \subseteq \mathcal{P}$, where $G(\bigcup_{X_i \in \mathcal{W}} X_i)$ is connected.

Lemma 2.7. *Let $U \subseteq V(G)$ be chosen such that $\{U\}$ is optimal for $G(U)$. Furthermore, let $\mathcal{P}'$ be an optimal partition for $G' = G(V \backslash U)$ with induced edge set $A^* \subseteq E(G')$. Then, for $\mathcal{W} \subseteq \mathcal{P}'$ that attains the minimal value of $|\mathcal{W}| - w(E(\mathcal{W} \cup U))$, the edge set $A^* \cup E(\mathcal{W} \cup U)$ is an optimal solution of the Potts problem for G.*

The optimality conditions from Lemma 2.7 are proven in [1] for the special case of $U = \{v\}$ for some node $v \in V(G)$.

3 Exact Algorithms

In this section we present an exact algorithm for solving the Potts problem (2.2). As it is based on that of Anglès d'Auriac et al., we first introduce the latter. Subsequently, we propose algorithmic modifications using the above-mentioned graph-theoretic results in order to decrease the running time on practical instances.

3.1 The Basic Exact Algorithm

The optimal cooperation algorithm proposed in [1] uses the optimality conditions given in Lemma 2.7, applied to node sets of cardinality 1.

It starts with a trivial solution for a subgraph $G(U)$ with $|U| = 1$. Iteratively, nodes are added to U one after one. In each step of the algorithm, an optimum solution on $G(U)$ is known. After having added a new node to U, an optimum solution on the new induced graph is computed from the former optimum by calculating a minimum $s - t$ cut in an associated network. If $U = V$, the solution at hand is optimum for the whole graph G.

Let us assume that problem (2.2) has been solved on $G(U)$ with $U \subset V(G)$, yielding $\mathcal{P}_U = \{X_1, \dots, X_k\}$ as optimal partition. Let $v \in V(G) \backslash U$. For maintaining optimality, we have to compute a set $W \subseteq U \cup \{v\}$, $v \in W$, that solves the problem

$$\min\{b(W) : \; W \subseteq U \cup \{v\}, \; v \in W\}. \tag{3.4}$$

In [1] it was shown that such a set W can be determined by the computation of a minimum cut in an associated directed network $D_{U,v}$. As the value of $b(W)$ does not depend on the sets $X_1, \dots, X_k$, the size of the network can be kept small by first contracting $X_1, \dots, X_k$ in G and constructing the network from the shrunk graph.

The flow of the original optimal cooperation algorithm proposed in [1] is displayed in Algorithm 3.1.

Algorithm 3.1. *(optimal cooperation algorithm from [1])*

```
Input: A graph $G = (V, E)$ with edge weights $w_e \in (0,1)$ for all edges $e \in E(G)$.
Output: An optimal partition $\mathcal{P}$ for $G$.
1. Set $U := \emptyset$ and $\mathcal{P} := \emptyset$.
2. Choose a vertex $v \in V \backslash U$.
3. Determine $W \subseteq U \cup \{v\}$ that solves (3.4) by computing a minimum $s - t$ cut in
   $D_{U,v}$.
4. Set $U := U \cup \{v\}$ and construct the new optimal partition $\mathcal{P}$.
5. Shrink the vertex set $W$ in $G$ and set $U := U/W$.
```

6. *If $U \neq V(G)$ go to Step 2.; else output the optimum partition $\mathcal{P}$ and STOP.*

In Algorithm 3.1 $|V(G)| - 1$ minimum cut problems are solved on graphs with $|U| + 3$ vertices and at most $2(|E(G(U \cup \{v\})| + |U| + 3)$ many arcs. If, e.g., the Goldberg-Tarjan algorithm [6] is used for the computation of the minimum $s - t$ cuts, the algorithm has a worst case running time of $O(|V(D_{U,v})|^2 \sqrt{|E(D_{U,v})|})$ in each of the $|V(G)| - 1$ iterations. Thus, Algorithm 3.1 has polynomial running time. However, its performance depends strongly on the size of the directed graphs $D_{U,v}$ and the number of minimum cut computations.

In the following subsection we present ideas for improving the performance of the algorithm, using the results of the previous sections.

3.2 Enhancement of the Basic Exact Algorithm

The results from the previous sections can be compiled to an improved algorithm that we present in Algorithm 3.2.

In Lemma 2.2 we have shown that there exists an optimum solution that does not contain an edge from a cut with weight at most 1. In the physics application, the edge weights are chosen randomly between 0 and 1, and the cardinality of the sets U defining these cuts tends to be small in practice. Moreover, often these node sets will contain one node only. Therefore, we determine all vertices $v \in V(G)$ with $w(\delta(v)) \leq 1$ and save them in a set $notU$. Subsequently, we apply the exact algorithm to the graph $G \backslash notU$. According to our experience, this reduces the number of minimum cut computations and the size of the networks $D_{U,v}$ considerably, leading to low running times.

Assume some $v \in V(G) \backslash U$ has been chosen in Step 2. If $w(\{(u,v) \in E(G) : u \in U\}) \leq 1$, an optimal solution of the Potts problem consists of $W = \{v\}$. Thus, we do not have to determine the corresponding minimum $s - t$ cut and can skip Steps 3. and 4. of the Algorithm 3.1.

Furthermore, subgraphs like cycles, cliques etc., satisfying the conditions from Lemma 2.4 are contracted. In order to be able to reconstruct the optimum partition from the shrunk graph, we assign a set $S(v)$ to the vertices $v \in V$ that save the original nodes shrunk into supernode v. These sets are updated throughout the algorithm.

Algorithm 3.2. *(improved algorithm)*

Input: *A graph $G = (V, E)$ with edge weights $w_e \in (0,1)$ for all edges $e \in E(G)$.*

1. *Compute the set $notU$ and delete all vertices $v \in notU$ from G.*
2. *Set $U := \emptyset$ and $\mathcal{P} := \emptyset$.*
3. *Choose $v \in V \backslash U$ and set $S(v) := \{v\}$.*
4. *If $w(\{(u,v) \in E(G) : u \in U\}) \leq 1$, set $U := U \cup \{v\}$ and $\mathcal{P} := \mathcal{P} \cup \{S(v)\}$ and proceed with Step 3.*
5. *While a subset $W \subseteq U \cup \{v\}$, $|W| > 1$, is found with $v \in W$ such that $\{W\}$ is an optimal partition for $G(W)$, do the following:*
   Set $U := U \backslash W$ and $\mathcal{P} := \mathcal{P} \backslash \bigcup_{u \in (W \backslash \{v\})} S(u)$. Contract W in G to a supernode v_W and set $S(v_W) := \bigcup_{u \in W} S(u)$. Identify $v := v_W$.
6. *Construct $D_{U,v}$ and determine $W \subseteq U \cup \{v\}$ that solves (3.4) by computing a minimum $s - t$ cut in $D_{U,v}$.*
   Then, update U, $\mathcal{P}$, the graph G, and v, analoguously to Step 5.

7. *While there exists an edge $e = (u, v) \in E(G)$ with $w_e \geq 1$ do:*
   If $u \notin U$, shrink e in G and set $S(v) := S(V) \cup \{u\}$. If $u \in U$, set $W := \{u, v\}$ and update U, $\mathcal{P}$, G, and v, analoguously to Step 5.
8. *If in Step 7. an edge $e = (u, v)$ was found with $u \notin U$, proceed with Step 4; else set $U := U \cup \{v\}$, $\mathcal{P} := \mathcal{P} \cup \{S(v)\}$, and proceed with Step 9.*
9. *If $U \neq V(G)$, go to Step 3.*
10. **Output:** *The optimum partition $\mathcal{P} := \mathcal{P} \cup \{\{v\} : v \in notU\}$.*

Why is Step 8. of the improved algorithm important? Assume W has been determined by computing a minimum $s - t$ cut in the network $D_{U,v}$. Then there possibly exist edges $e = (v, u)$ in G/W with $w_e = 1$ and $u \in U$. In this case, the partition we obtain after shrinking these edges is still optimal for the graph $G(U \cup \{v\})$. However, there possibly exist edges $e = (v, u)$ in G/W with $w_e \geq 1$ and $u \notin U$. Then, after contracting these edges, in general $\mathcal{P} \cup \{S(v)\}$ is not an optimal partition for $G(U \cup \{v\})$. Therefore, we have to repeat the iteration by using vertex v.

In Step 5. of the Algorithm 3.2 we search iteratively for a subset $W \subseteq U \cup \{v\}$ with $|W| > 1$ and $v \in W$ such that $\{W\}$ is optimal for $G(W)$. This also includes the search for edges with a weight larger or equal one. In the following, we focus on cycles and subgraphs F_2. If we succeed in our search, contracting the corresponding set decreases the size of the networks and the number of minimum cut computations in the subsequent iterations.

Next we answer the question, how to find a cycle $C = (V_C, E_C)$ in a graph G satisfying $\sum_{e \in E_C} w_e \geq |V_C| - 1$ algorithmically. Let G^- denote a copy of G, where edge weights are chosen as $w_e^- := 1 - w_e$ for all $e \in E(G)$. Then the condition to be satisfied can be expressed as $\sum_{e \in E_C} w_e^- \leq 1$. Cycles satisfying the latter can be computed by shortest-path calculations in G^- as follows. For each edge $e = (v, u) \in E(G)$, we temporarily delete e from G^- and search for a shortest path $P = (V_P, E_P)$ in G^- from v to u. If the weight of the path does not exceed $1 - w_e^-$, P together with e forms a cycle satisfying the condition. Then V_P is contracted in Step 5. of Algorithm 3.2. This method is an exact cycle search, i.e., if there exists a cycle satisfying the condition, the algorithm finds it. In addition to this, we also use fast heuristics. In practice it turned out that heuristics and simple enumerative checks for triangles in the graph suffice to find candidate cycles.

In the next section we present some experimental results and compare the performance of the basic with that of the improved algorithm.

4 Computational Results

We implemented both the original algorithm by Anglès d'Auriac et al. and our modifications within the same framework, using the OGDF library [15]. For the minimum cut computations, we used a fast implementation of the Goldberg-Tarjan algorithm [16]. The runs were performed on an Intel Celeron machine with 1.86 GHz.

Before starting Algorithm 3.2, we ran a heuristic for the computation of problem (1.1). This heuristic is constructed simply by skipping the minimum cut computations in Step 6. of Algorithm 3.2. In order to solve the problem exactly, the heuristics is followed by the improved exact algorithm. Furthermore, in Step 5. of the latter, we used fast heuristics for the cycle search and an enumerative algorithm for the $F_2 =$

$G(W)$ subgraphs consisting of two triangles sharing an edge for which $\{W\}$ is optimal for F_2.

As we are not aware of other experimental results presented in the literature, we focused on the physics application. For the latter, instances defined on regular grid graphs in d dimensions with weights chosen according to some probability distribution are very relevant. In order to be able to compare our results with the algorithm implemented in [1], we used the same test bed. We studied two-dimensional square grid graphs with randomly chosen weights that can take only two different values, $w_1 \neq w_2$, where $p\%$ of the edges have weight w_1. Furthermore, the weights satisfy $w_1 + w_2 = 1$. We vary the size L of the $L \times L$ grid, w_1 and p. As the weights are randomly chosen, we always report averages over 20 different instances of the same class.

We compare the behaviour of the different algorithms for $L = 128$ and different values of w_1 and p in Table 1 and Table 2. We report the CPU times in seconds, the number of minimum cut computations, and the maximum size of U, for which a minimum cut is computed. For the choice of $w_1 = 0.2$, the cycle search heuristics

p	Improved Method			Algorithm 3.1		
	# mincut comp.	maximal U	CPU time	# mincut comp.	maximal U	CPU time
80	6.30	64.55	0.48	16383	16301.80	138.06
70	210.65	1566.00	4.50	16383	15979.70	145.93
60	1765.30	9983.85	19.07	16383	14979.65	153.05
50	2364.10	5738.45	19.03	16383	7635.80	115.61
40	265.05	183.70	4.93	16383	907.30	21.43
30	3.90	13.55	3.15	16383	218.45	14.10
20	0.25	0.55	2.14	16383	133.40	12.50

Table 1. Results for $L = 128$ and $w_1 = 0.2$. CPU times are given in seconds.

is very often successful. Moreover, the heuristic for large p often solves the problem exactly. Also for smaller p the heuristic helps reducing the running time. We note that the basic Algorithm 3.1 always computes $|V| - 1$ minimum cuts, independently of the distribution of the edge weights. In contrast, in the improved algorithm the number depends on the choice of w_1 and p.

For $w_1 = 0.4$, the heuristic has only a running time of about 0.2 seconds, but it is never successful in reducing the size of the graph. Furthermore, the cycle search algorithms contribute less than for $w_1 = 0.2$. However, for $w_1 = 0.4$ the search for the subgraphs F_2 and Step 4. of Algorithm 3.2 are often successful. We compare the running times in seconds in Table 2. Clearly, the running times can drastically be reduced by the above-mentioned graph-theoretic considerations.

p	80	70	60	50	40	30	20
Improved Method	9.24	17.26	28.55	74.95	8.09	3.38	2.89
Algorithm 3.1	142.64	149.96	150.52	279.99	15.08	13.09	12.82

Table 2. Solution times in seconds for $L = 128$ and $w_1 = 0.4$

In 2002, Anglès d'Auriac et al. presented average running times for $L \times L$ grid graphs [1] with $w_1 + w_2 = 1$, using a Pentium III processor with 800 MHz. Their program needed on average 1.5 hours for a 128^2 grid and one day of CPU time for a grid of size 256^2. Our implementation of the basic algorithm needs roughly half an hour on average for $L = 256$, whereas the improved method only takes ca. 6 minutes. We note that the machine we used is considerably faster, and so we cannot easily compare our running times with those reported in [1].

5 Conclusions

In this work, we presented a fast algorithm for the problem of optimum cooperation. By an intensive study of the underlying graph-theoretic problem, the running time of the solution algorithm can considerably be reduced. It appears plausible that this trend can be further strengthened by investing knowledge on further graph structures, in the flavor of Lemma 2.4.

Acknowledgments

We are grateful to Andrea Wagner for significant help with the implementation of the algorithm. We thank Heiko Rieger for valuable remarks on the physics aspects of the problem.

References

1. J.-Ch. Anglès d'Auriac, F. Iglói, M. Preissmann, and A. Sebö, *Optimal cooperation and submodularity for computing Potts' partition functions with a large number of states*, J. Phys. A: Math. Gen. 35 (2002) pp. 6973-6983
2. M. Baïou, F. Barahona, R. Mahjoub, *Separation of partition inequalities*, Math. of Operations Research, vol. 25, no. 2 (2000) pp. 243-254
3. F. Barahona, *Separating from the dominant of the spanning tree polytope*, Operations Research Letters, vol. 12 (1992) pp. 201-203
4. S. T. McCormick, *Submodular Function Minimization*, In: K. Aardal et al., Eds.,*Discrete Optimization: Handbooks in Operations Research and Management Science, vol.12* , Elsevier (2005) pp. 321-391
5. W. H. Cunningham, *Optimal Attack and Reinforcement of a Network*, Journal of the Association for Computing Machinery, vol. 32, no. 3 (1985) pp. 549-561
6. A.V. Goldberg, R. E. Tarjan, *A new approach to the maximum-flow problem*, Journal of the ACM, vol 35, no. 4 (1988), pp. 921-940.
7. S. Iwata, L. Fleischer, and S. Fujishige, *A combinatorial strongly polynomial algorithm for minimizing submodular functions*, Journal of the ACM **48** No. 4, 761-777 (2001).
8. S. Fujishige, *Submodular Functions and Optimization*, Annals of Discrete Mathematics Vol.58, Elsevier, Amsterdam, 2005
9. M. Grötschel, L. Lovász, A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 169-197 (1981).
10. M. Grötschel, L. Lovász, A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer Verlag (1988).
11. A.K. Hartmann, H. Rieger, *Optimization Algorithms in Physics*, Wiley-VHC, Berlin, 2002
12. A. K. Hartmann, H. Rieger, *New Optimization Algorithms in Physics*, Wiley-VHC, Berlin, 2004
13. R. Juhasz, H. Rieger, F. Iglói, *The random-bond Potts model in the large-q limit*, Phys. Rev. E, **64**, 056122 (2001).
14. M. Preissmann, A. Sebö, *Graphic submodular function minimization: an graphic approach and applications* , unpublished

15. Open Graph Drawing Framework, `www.ogdf.net`
16. M. Jünger, G. Rinaldi, S. Thienel, *Practical Performance of Efficient Minimum Cut Algorithms*, Algorithmica, 26, 172-195, (2000).
17. A. Schrijver, J. Combinatorial Theory, B **80**, *A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time* 346–355 (2000).

Appendix

In this appendix, we state the most relevant proofs that we omitted in the extended abstract due to space restrictions.

Proof (Lemma 2.1). The supermodularity of the Potts function implies

$$f_{G,w}(A \cup A_1) \geq f_{G,w}(A) + f_{G,w}(A_1) - f_{G,w}(A \cap A_1) \tag{5.5}$$

for all edge sets $A \subseteq E(G)$. Furthermore, since A_1 and $A_1 \cap A$ are subsets of $E(G_1)$ it holds

$$f_{G,w}(A_1) = f_{G_1,w}(A_1) + |U| \geq f_{G_1,w}(A_1 \cap A) + |U| = f_{G,w}(A \cap A_1). \tag{5.6}$$

These inequalities imply $f_{G,w}(A \cup A_1) \geq f_{G,w}(A)$ for all edge sets $A \subseteq E(G)$. Thus, there exists some optimum solution A_1^* with $A_1^* \supseteq A_1$.

Similarly, there exists some optimum solution A_2^* with $A_2^* \supseteq A_2$. Using the equality $f_{G,w}(A_1^*) = f_{G,w}(A_2^*)$ and the supermodularity of the Potts function $f_{G,w}$, it is easy to prove that $A^* = A_1^* \cup A_2^*$ is an optimal solution for G. The inclusion $A^* \supseteq A_1 \cup A_2$ is obvious. □

Proof (Lemma 2.2). Let be given an optimal solution $A \subseteq E(G)$ of the Potts problem with $A \cap \delta(U) \neq \emptyset$. Then it holds $c_G(A \backslash \delta(U)) \geq c_G(A) + 1$. Furthermore, $w(A \cap \delta(U)) \leq w(\delta(U)) \leq 1$. Hence, it holds

$$\begin{aligned} f_{G,w}(A \backslash \delta(U)) &= c_G(A \backslash \delta(U)) + w(A) - w(A \cap \delta(U)) \\ &\geq c_G(A) + w(A) + 1 - w(A \cap \delta(U)) \\ &\geq f_{G,w}(A), \end{aligned} \tag{5.7}$$

i.e., $A^* := A \backslash \delta(U)$ is also optimal and $\delta(U) \cap A^* = \emptyset$. □

Proof (Lemma 2.3). Let $\mathcal{P}$ resp. $\mathcal{P}^*$ be optimal partitions for G resp. G/U. Then it holds $F_{G/U,w}(\mathcal{P}^*) \geq F_{G/U,w}(\mathcal{P}')$. Decontracting the set U in $\mathcal{P}^*$, we obtain a partition $\mathcal{P}^{**}$ for G with

$$F_{G,w}(\mathcal{P}^{**}) = F_{G/U,w}(\mathcal{P}^*) + w(U) \geq F_{G/U,w}(\mathcal{P}') + w(U) = F_{G,w}(\mathcal{P}). \tag{5.8}$$

Consequently, the optimality of $\mathcal{P}$ implies $F_{G/U,w}(\mathcal{P}^*) = F_{G/U,w}(\mathcal{P}')$, i.e., $\mathcal{P}'$ is an optimal partition for G/U.

Now we assume that $\mathcal{P}'$ an optimal partition for G/U. Then Lemma 2.1 and the optimality of $\tilde{\mathcal{P}} = \{U\}$ for $G(U)$ imply that there exists an optimal partition $\mathcal{P}^{**} = \{P_1^{**}, \dots, P_l^{**}\}$ of G with $U \subseteq P_l^{**}$. Hence, $\mathcal{P}^* = \{P_1^{**}, \dots, P_{l-1}^{**}, P_l^{**}/U\}$ is a partition of G/U. Using the optimality of $\mathcal{P}'$ we obtain

$$F_{G,w}(\mathcal{P}^{**}) = F_{G/U,w}(\mathcal{P}^*) + w(U) \leq F_{G/U,w}(\mathcal{P}') + w(U) = F_{G,w}(\mathcal{P}), \tag{5.9}$$

i.e., $\mathcal{P}$ is optimal for G. □

Before we can proceed with the proof of Lemma 2.4 we need to prove the following lemma.

Lemma 5.1. *Let G be a graph with an optimal partition $\mathcal{P} = \{X_1, \dots, X_k\}$. Further, let $I \subset \{1, \dots, k\}$ be a nonempty index set inducing a subgraph $G' = G(\bigcup_{i \in I} X_i)$ of G. Then the partition $\mathcal{P}' = \{X_i : i \in I\}$ is optimal for G'.*

Proof (Lemma 5.1). Assume that the lemma is not valid. Then there exists a partition $\bar{\mathcal{P}}$ for G' with $F_{G',w}(\bar{\mathcal{P}}) > F_{G',w}(\mathcal{P}')$. Consequently,

$$\begin{aligned} F_{G,w}(\mathcal{P}) = |\mathcal{P}| + \sum_{i=1}^{k} \sum_{x,y \in X_i} w(x,y) &= (k - |I|) + \sum_{i \notin I} \sum_{x,y \in X_i} w(x,y) + F_{G',w}(\mathcal{P}') \\ &< (k - |I|) + \sum_{i \notin I} \sum_{x,y \in X_i} w(x,y) + F_{G',w}(\bar{\mathcal{P}}) = F_{G,w}(\mathcal{P}'') \qquad (5.10) \end{aligned}$$

with the partition $\mathcal{P}'' = \{X_i : i \notin I\} \cup \bar{\mathcal{P}}$ for G, which is a contradiction to the optimality of $\mathcal{P}$. □

Proof (Lemma 2.4). We will prove Lemma 2.4 in detail only for cycles. For complete graphs the lemma can be proven using induction and Lemma 5.1. The criterion for the graph F_2 results from Lemma 5.1 and considerations for forests and triangles.

Before we can start with the investigation of cycles, we consider forests. We first prove the following statement:

Let $G = (V, E)$ be a forest with edge weights $w_e \in (0, 1)$ for all edges $e \in E(G)$. Then $A^* = \emptyset$ is the unique optimal solution of the Potts problem.

In fact, if G is a forest, for each nonempty sets $A \subseteq E(G)$ it holds

$$f_{G,w}(A) = c_G(A) + \sum_{e \in A} \underbrace{w_e}_{<1} < c_G(A) + |A| = |V(G)| = f_{G,w}(\emptyset), \qquad (5.11)$$

since for each subset $A \subseteq E(G)$ the graph $G(A) = (V, A)$ is a forest with $|V(G)|$ vertices. Consequently, $A^* = \emptyset$ is an optimal solution for G.

Now, let us consider cycles. Let $A^* \subseteq E(C^n)$ be an optimal solution of the Potts problem (2.2). Assume it is $0 < |A^*| < n$. Then Lemma 5.1 implies that A^* is also an optimal solution for the subgraph $G' = G(A^*)$. The graph $G' = G(A^*)$ is a forest because of $|A^*| < n$. But then $0 < |A^*|$ is a contradiction to the previous result on forests.

Thus it holds $A^* = \emptyset$ or $A^* = E(C^n)$. This implies the statement of the lemma since the inequality $\sum\limits_{e \in E(C^n)} w_e \geq n - 1$ is equivalent to the inequality

$$f_{C^n,w}(\emptyset) = n \leq \sum_{e \in E(C^n)} w(e) + 1 = f_{C^n,w}(E(C^n)). \qquad (5.12)$$

□

Proof (Lemma 2.5). Assume that $\mathcal{P}^* = \{U\}$ is not optimal for $G' = G(U)$ with edge weigths w_e chosen as $w_e \geq w'_e$ for all edges $e \in E(G)$, i.e., an optimal partition $\mathcal{P} = \{P_1, \dots, P_k\}$ of $G' = G(U)$ exists with $F_{G',w}(\mathcal{P}) > F_{G',w}(\mathcal{P}^*)$. Then it holds $E(\mathcal{P}) \subseteq E(\mathcal{P}^*)$. Let $E' = E(\mathcal{P}^*) \backslash E(\mathcal{P})$. Then $F_{G',w}(\mathcal{P}) > F_{G',w}(\mathcal{P}^*)$ implies

$$w(\mathcal{P}^*) + 1 < w(\mathcal{P}) + k \qquad (5.13)$$

$$w'(E') \leq w(E') = w(\mathcal{P}^*) - w(\mathcal{P}) < k - 1 \qquad (5.14)$$

$$F_{G',w'}(\mathcal{P}^*) = w'(\mathcal{P}^*) + 1 < w(\mathcal{P}) + k = F_{G',w'}(\mathcal{P}^*). \qquad (5.15)$$

This is a contradiction to the optimality of $\mathcal{P}^*$ for the edge weights w'_e. $\square$

Proof (Lemma 2.7). Let $U \subseteq V(G)$. Let $\{U\}$ resp. $\mathcal{P}' = \{X_1, \dots, X_k\}$ be optimal for $G(U)$ resp. $G' = G(V \backslash U)$. Because of Lemma 2.6 there exists a set $\mathcal{W}^* \subseteq \mathcal{P}$ with $\mathcal{P} = \{X_1, \dots, X_k, U\}$ such that $(\mathcal{P} \backslash \mathcal{W}^*) \cup \{\{\cup X : X \in \mathcal{W}^*\}\}$ is an optimal partition for G. It induces an edge set $A^* \cup E(\mathcal{W}^*)$. We only have to prove that w.l.o.g. $U \in \mathcal{W}^*$ can be assumed.

Suppose $U \notin \mathcal{W}^*$. Then it holds $|\mathcal{W}^*| - 1 - w(E(\mathcal{W}^*)) \geq 0$ because of the optimality of $\mathcal{P}'$ for $G(V \backslash U)$. Using the optimality of $(\mathcal{P} \backslash \mathcal{W}^*) \cup \{\{\cup X_i : X_i \in \mathcal{W}^*\}\}$ for G and Lemma 2.6 we obtain $|\mathcal{W}^*| - 1 - w(E(\mathcal{W}^*)) = 0$. But then we can also choose $\mathcal{W}^* = \{U\}$ to obtain an optimal partition. $\square$