BPLS: Cutset-Driven Local Search For MPE and Improved Bounds for Minimal Cutsets in Grids

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science

by

Alon Milchgrub

Supervised by

Professor Rina Dechter and
Professor Amir Globerson

The School of Computer Science and Engineering
Hebrew University of Jerusalem, Israel

Winter 2014

BPLS: Cutset-Driven Local Search For MPE and Improved Bounds for Minimal Cutsets in Grids

Abstract

BPLS: Cutset-Driven Local Search For MPE and Improved Bounds for Minimal Cutsets in Grids

by

Alon Milchgrub
Master of Science in Computer Science

The problem of finding an optimum of a multivariate function described as a sum of potentials over (small) subsets of variables is one of fundamental interest both in probabilistic inference and other fields. In this thesis we present a cycle-cutset driven stochastic local search algorithm which approximates the optimum of sums of unary and binary potentials, called Belief Propagation Local Search or BPLS. We evaluate empirically the effects of different components of BPLS on its performance and present the results of extensive and comprehensive experiments conducted on several problem sets.

We further suggest several novel heuristics designed to improve the exploration of the search space based on previous results and more detailed analysis of the energy function. Some of the heuristics and observations made are general, and may be applied in other local search algorithms and in other contexts. We present experimental results supporting the contribution of these heuristics. Finally, we compare the performance of the leading variants of BPLS to the state-of-the-art GLS$^+$ and against a hybrid. We show that in general the performance of BPLS is on-par with GLS$^+$ and that it significantly outperforms GLS$^+$ on the CSP problem set. Our results are significant because for the past decade, GLS$^+$ is well established as the best stochastic local search for MPE. Moreover, BPLS reaches strong local optima in the limit (conditionally optimal on every tree), and as such provides an effective and convergent algorithm alternative to min-sum BP schemes.

In the second part of this thesis we explore the notion of tree inducing cycle-cutset and present novel theoretical results. We prove that in grids of any size there exists a minimal cycle-cutset whose complement induces a single connected tree. More generally, any cycle-cutset in a grid can be transformed to a tree-inducing cycle-cutset, no bigger than the original one in a series of steps from a given cutset to another. We use this result to improve the known lower bounds on the size of a minimal cycle-cutset in certain cases of grids, thus equating the lower bound to the known upper bound.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Rina Dechter for suggesting me the idea of BPLS for my project in her class, and for embracing my original work and taking me under her wing, thus growing the class project to a paper and eventually to this thesis.
I would like to thank Amir Globerson for encouraging me to explore the theoretical aspects of my work.
A special thanks should be given to Raanan Fattal, which as a result of our joint work and his support I got to feel like a member of the HUJI CS department at least to some extent.

Finally, my thanks go to my beloved mother and Gal for their care, ongoing support and contribution in submitting this thesis in my name in my absence.

Chapter 1

The Energy Minimization Problem

1.1 Introduction

The problem of optimizing discrete multivariate functions or *energy functions* described as a sum of potentials on (small) subsets of variables is one of fundamental importance and interest in a wide variety of fields. In the context of probabilistic graphical models, instances of these problems can be found in the form of *most probable explanation* (MPE) problems. By this paradigm uncertainty is modeled by using a finite set of discrete random variables and a corresponding set of conditional probability tables (CPT) describing the probabilistic relations between them. The resulting energy function is related to the probability of the random variables attaining a certain assignment. Finding a maximum of this energy function translates to finding an assignment of maximum probability given some partial assignment as evidence.

In the context of image processing and computer vision these problems emerge in many cases where each pixel in an image may attain one of a finite set of values. In these cases, the local interaction of small groups - in many cases, pairs - of pixels is described by potentials over the interacting pixels. An energy function composed of these potentials is then formulated and optimized in order to enforce a global consistency constraint on the entire image.

In light of the significance of this problem in many applications, it was extensively studied although it is $\mathcal{NP}$-hard to solve in general. In addition to a variety of complete search algorithms[18, 5], other means of tackling the problem were suggested including approximation algorithms[23, 11, 21] as well as algorithms designed to solve specific classes of instances of this problem[4].

When solving an energy optimization problem, it is usually useful to represent the energy function to be optimized by a graph called the *primal graph*, in which every variable is represented by a vertex and two vertices are connected by an edge iff the two matching variables appear in a single potential. The characteristics of the primal graph provide us with much information about the nature of the problem and the problem's complexity is mainly governed by the structure of the graph. For example, energy functions whose primal graph is acyclic can be efficiently and exactly optimized using a *belief propagation* (BP) algorithm[22]. In case the graph contains cycles, the values of some variables can be set, thus effectively eliminating their matching vertices from the graph until all cycles are opened. At which point, BP can be run on the remainder of the variables, thus finding an optimum of the function given the values of the set variables. This method of generating acyclic graphs by instantiating a subset of the variables is known as *cycle-cutset conditioning*[8].

In this work, we concentrate on the optimization of functions described as a sum of unary and binary potentials only. The approach we suggest, *stochastic tree-based local search* or BPLS, is based on iteratively improving the current assignment by generating a different cycle-cutset in every iteration and finding the exact optimum on the rest of the variables, given the values of the cutset variables using BP. This is sufficient in order to produce a local search algorithm progressing in blocks of variables. However, in order

to accelerate the process, a subsidiary local search algorithm is run on the current cutset variables, while the forest variables are optimized using BP. Remarkably, as we show, BPLS is guaranteed to provide strong local optima at the limit, thus yielding an affective alternatives to min-sum loopy BP schemes, that often do not converge.
We build on earlier work by [24], where the ideas of traversal from one cutset to another and restriction of basic stochastic local search to cycle-cutsets variables were introduced. These ideas were never evaluated empirically, except for a variant focusing on solving the constraint satisfaction task [13] employing the idea of separately improving the assignments of cutset variables and forest variables, but operating only with a constant cutset. In this context our contribution is in identifying and empirically exploring a new such stochastic scheme.

1.2 Problem Definition and Preliminaries

Definition 1 (Energy Minimization Problem)**.** let $\bar{X} = X_1, \ldots, X_N$ be a set of variables over a finite domain $\mathcal{D}$ of size k, let $\varphi_i : \mathcal{D} \to \mathbb{R}$ for $i \in \{1, \ldots, N\}$ be unary potentials, and let $\psi_{i,j} : \mathcal{D}^2 \to \mathbb{R}$ for a subset of pairs $E \subseteq \{\{i,j\} : 1 \leq i < j \leq N\}$ be binary potentials, then these potentials define an *energy function* of the following form

$$\mathcal{E}\left(\bar{X}\right) = \sum_i \varphi_i\left(X_i\right) + \sum_{\{i,j\} \in E} \psi_{i,j}\left(X_i, X_j\right)$$

and the *energy minimization problem* is finding

$$\bar{x}^* = \arg\min_{\bar{x}} \mathcal{E}\left(\bar{x}\right) = \arg\min_{\bar{x}} \sum_i \varphi_i\left(x_i\right) + \sum_{\{i,j\} \in E} \psi_{i,j}\left(x_i, x_j\right)$$

Clearly min-sum problems are equivalent to max-sum, (and to max-product and min-product) and in our empirical evaluation we focus on benchmark requiring solving the max-sum problem. In the context of max-sum, the "*Energy*" is also referred to as "*Goodness*" indicating that more is better.

Definition 2 (Primal Graph)**.** Given an instance of the energy minimization problem, the *primal graph* of the problem is a graph where every variable x_i is assigned a vertex, and two vertices x_i and x_j are connected if there exists a potential $\psi_{i,j}$.

Definition 3 (Cycle-Cutset)**.** Let $G = (V, E)$ be an undirected graph. A *cycle-cutset* in G is a subset C of V, such that the graph induced on $V' = V \backslash C$ is acyclic, i.e. a forest.

Definition 4 (Parent, Children and Neighbors)**.** Let $G = (V, E)$ be a directed graph such that the out-degree of every vertex $v \in V$ is at most 1, and let $v \in V$ be a vertex, if $v$points to another vertex u we say that u is the *parent* of v and denote $u = pa\left(v\right)$. All vertices $w_1, \ldots, w_k$ point to v are called the *children* of v and are denoted as $ch(v)$, that is for every $v \in V$

$$ch\left(v\right) = \{w \,:\, (w, v) \in E\}$$

all vertices neighboring with v are called the *neighbors* of v and are denoted $nb(v)$, that is for every $v \in V$

$$nb(v) = \{u : (u,v) \in E \wedge (v,u) \in E\} = ch(v) \cup \{pa(v)\}$$

1.3 Background

1.3.1 Belief Propagation

If the primal graph of an energy function to be minimized is acyclic, the problem can be solved efficiently and exactly using an algorithm called *belief propagation* (BP) [22, 3]. The paradigm of belief propagation can be applied to many different problems and can have numerous manifestations. One of its variants used to solve the energy minimization problems operates as follows:

Given an energy function with an acyclic primal graph, i.e. a forest, a root is assigned for every tree in the forest and all edges are directed toward it, thus defining for every vertex v other than the root a parent $pa(v)$ and defining for every vertex v a set of children $ch(v)$ which may be empty. Then, starting from the leaves and going upward along each tree, every non-root vertex v calculates for every possible value $x_{pa(v)}$ of its parent the functions ρ_v and μ_v, which are defined recursively as follows.

$$\rho_v\left(X_v, X_{pa(v)}\right) = \varphi_v(X_v) + \psi_{v,pa(v)}\left(X_v, X_{pa(v)}\right) + \sum_{u \in ch(v)} \mu_u(X_v) \tag{1.1}$$

$$\mu_v\left(X_{pa(v)}\right) = \min_{x_v} \rho_v\left(x_v, X_{pa(v)}\right) \tag{1.2}$$

The function μ_v is then passed from v to its parent $pa(v)$ as a "message" to continue the calculation.

Let r be a root in the primal graph, then the definition of ρ_v can be naturally extended to r as follows:

$$\begin{aligned} \rho_r(X_r) &= \varphi_r(X_r) + \sum_{u \in ch(r)} \mu_u(X_r) \\ \mu_r &= \min_{x_r} \rho_r(x_r) \end{aligned}$$

It obvious that for every vertex v the value of function $\mu_v\left(X_{pa(v)}\right)$ can be calculated for a single value $X_{pa(v)} = x_{pa(v)}$ in time linear in the size k of the domain of X_v, given the functions μ_u of all his children $u \in ch(v)$. Therefore, since by assumption all variables share the same domain $\mathcal{D}$ of size k, calculating the function μ_v can be done in time $\mathcal{O}(k^2)$, and calculating μ_v for all $v \in V$ can be done in time $\mathcal{O}(Nk^2)$ (where N is the number of vertices\variables).

Let v be a vertex and let $\mathcal{E}_v$ be the energy function obtained from $\mathcal{E}$ when restricted to only potentials whose variables are descendents of v in the directed primal graph in addition to $\psi_{v,pa(v)}$ if v is not a root. Then, the following two claims can be proven by induction:

Claim 5. If v is not the root of a tree in the forest, then $\mu_v\left(x_{pa(v)}\right)$ is the minimum of $\mathcal{E}_v$ given that $X_{pa(v)} = x_{pa(v)}$. Otherwise, i.e. if $v$is the root of a tree, then μ_v is the minimum of $\mathcal{E}_v$.

Claim 6. If v is not a root of a tree, then the value of $\mu_v\left(x_{pa(v)}\right)$ is attainable by an assignment to $\mathcal{E}_v$ in which $X_v = \arg\min_{x_v} \rho_v\left(x_v\right)$ and $X_{pa(v)} = x_{pa(v)}$.

Otherwise, the value of μ_v is attainable by an assignment to $\mathcal{E}_v$ in which $X_v = \arg\min_{x_v} \rho_v\left(x_v\right)$.

Once the function μ_u of every child u of a root is calculated (and passed to it), the message passing is redirected downward going from the root to the leaves: The root assumes a value as given by equation 1.3 and every other vertex v assumes a value depending on the value of its parent as given by Equation 1.4.

$$X_r = \arg\min_{x_r} \rho_r\left(x_r\right) \tag{1.3}$$

$$X_v = \arg\min_{x_v} \rho_v\left(x_v, x_{pa(v)}\right) \tag{1.4}$$

Considering claims 5 and 6 it is easy to see that an assignment in which a root variable X_r is set according to equation 1.3 can be extended to an assignment which achieves the minimum of $\mathcal{E}_r$. In addition, it follows that the way to properly extend this assignment is given by equation 1.4. That is, we get that the aforementioned procedure results in an assignment that achieves the minimum energy on the sub-problem induced on a single connected component of the primal graph. All in all, since sub-problems corresponding to disjoint components of the primal graph are independent, we get that BP solves the minimization problem exactly and in polynomial time.

1.3.2 Cycle-Cutset Conditioning

If the primal graph of an energy function contains cycles, a cycle-cutset can be found and the optimal assignment to the remaining forest variables can be efficiently found *given* an assignment to the cutset variables. It is easy to see that by exhaustively searching the optimal assignment to the forest variables given all assignments to the cutset's, one is able to find the optimal assignment to the entire problem in a method known as "*cutset-conditioning*"[23, 6]. It can also be easily seen that such a method is exponential in the size of the cutset found. Since finding a cycle-cutset of minimum cardinality given a graph was previously proven to be $\mathcal{NP}$-complete for general graphs[12], and since such a minimum cutset may be considerably big (for example in grids the minimum cutset is

of order of magnitude of a third of the graph, see [16, 17]) this method is not feasible in many real-life problems of even moderate size.

1.3.3 GLS and GLS$^+$

In light of the similarity of the energy minimization problem to MAX-SAT, [21] presented a reduction from energy minimization problems to MAX-SAT problems and suggests adapting MAX-SAT approximation algorithms to approximating energy minimization problems. The experimental results of [21] indicate a clear dominance of *Guided Local Search* (GLS) [27, 20] over *Discrete Lagrangian Multipliers* (DLM) [28] and *Greedy + Stochastic Simulation* [14] in solving synthetic problems and significant superiority of GLS on real-world problems over the competition. As the name implies GLS is a local search algorithm, thus attempting find an optimal assignment by transitioning between neighboring states in the search space, i.e. by iteratively improving the assignment by making small modifications to it. However, in order to drive the search away from local optima, the objective function is dynamically altered in GLS by adding to it penalties associated with *solution components*. More specifically, for every potential ϕ over the variables $\bar{X}_\phi$ every assignment $\bar{x}_\phi$ to $\bar{X}_\phi$ is considered a solution component and is assigned a penalty $\lambda_\phi(\bar{x}_\phi) \equiv \lambda_\phi(\bar{x})$. The evaluation function to be minimized in GLS is $g\left(\bar{X}\right) = \sum_\phi \lambda_\phi\left(\bar{X}\right)$. Once a local optimum is reached some of the penalties are incremented. The penalties incremented are those of partial assignments who apply in the current assignment and who achieve the maximum of a *utility function*, which depends on the value of the potential and the current value of the penalty. In addition, once in a certain number of local optima reached, all the penalties are multiplied by a smoothing factor $\rho < 1$ (in general ρ could be set equal to 1, but experiments have shown that doing so impairs the results achieved by the algorithm [11]).

Though the results of GLS in [21] were favorable in respect to other local search algorithms, they could still not surpass those of traditionally used systematic search algorithms such as *Branch and Bound* (B&B) [19]. However, considering the promising results [11] have taken several steps in order to improve the performance of GLS in an algorithm named GLS$^+$. First of all, the problem is reduced using a *variable elimination* [7] until a potential with more than a certain number of entries is produced. More over, in contrast to GLS which is initialized randomly, GLS$^+$ is initialized using a Mini-Bucket variant named MB-w(10^5) [9]. Thirdly, in order to induce faster convergence to the solution the potentials themselves (and not only the penalties) were logarithmically incorporated in the evaluation function, making the evaluation function to be **maximized** $g\left(\bar{X}\right) = log\left[\phi\left(\bar{X}_\phi\right)\right] - w \times \sum_\phi \lambda_\phi\left(\bar{X}\right)$, where w is a weighting factor. Furthermore, the smoothing factor ρ was raised from 0.8 as in [21] to 0.999. Finally, GLS$^+$ makes use of two novel caching schemes: one which stores at every step the score achieved when flipping any single variable to any of its values, and another one that stores all the variables who flipping their value to any other value leads to improvement of the score. In the

experiments of [11] GLS$^+$ was able to outperform the at-the-time state-of-the-art B&B algorithms by reaching the optimal solution in an order of magnitude faster. This has established GLS$^+$ as the state-of-the-art in respect to solving general energy minimization problems, and it has kept its status during countless experiments since.

1.3.4 Hopfield Model

A Hopfield network[10] is a form of an artificial neural network with binary neurons, that is every neuron can attain one of two possible values (usually 1 and -1, or 1 and 0). Hopfield networks are used in order to memorize certain assignments to the neurons by encoding these assignments as minimums of an energy function based on weights assigned to the interaction between pairs of neuron, i.e. the edges between them and thresholds assigned to each neuron. More specifically, for a network with N neuron, the assignments stored are minimums of the following energy function:

$$E\left(\bar{s}\right) = \sum_{i} \vartheta_i s_i - \frac{1}{2} \sum_{i \neq, j} w_{ij} s_i s_j$$

where s_i is the value of the i'th neuron, ϑ_i is the threshold of i'th neuron and the weight $w_{i,j}$ of the edge between the i'th and the j'th neurons is symmetric, i.e. $w_{i,j} = w_{j,i}$. Therefore, in order to recover a stored memory, one needs to solve an energy minimization problem of the form we are interested in. In the Hopfield model a minimum of this energy function is found using a local search algorithm based on a simple activation function: When updated, the i'th neuron attains the value s_i based on the following rule:

$$s_i = \begin{cases} 1 & if \ \sum_{j:j \neq i} w_{ij} s_j \geq \vartheta_i \\ 0 & otherwise \end{cases}$$

It should be noted that changing the value of the i'th neuron affects only terms associated with it in the energy function, that is the threshold ϑ_i and the weights $w_{i,j}$ and $w_{j,i}$ for $j \neq i$. Specifically, when the i'th neuron changes it's value from s_i to s'_i (and accordingly the energy changes from E to E') the change in the energy is

$$\begin{aligned} \Delta E &= E' - E = \vartheta_i \left(s'_i - s_i\right) - \frac{1}{2} \left(s'_i - s_i\right) \sum_{j:j \neq i} w_{ij} s_j - \frac{1}{2} \left(s'_i - s_i\right) \sum_{j:j \neq i} w_{ji} s_j \\ &= \left(s'_i - s_i\right) \left[\vartheta_i - \sum_{j:j \neq i} w_{ij} s_j\right] \end{aligned}$$

where the last equality is based on the symmetry of the weights.

We see that if $\sum_{j:j \neq i} w_{ij} s_j \geq \vartheta_i$ then $\vartheta_i - \sum_{j:j \neq i} w_{ij} s_j \leq 0$ and therefore the maximal **decrease** in the energy will be attained when the difference $s'_i - s_i$ is biggest, that is if $s'_i = 1$. In the other direction, if $\sum_{j:j \neq i} w_{ij} s_j < \vartheta_i$ then $\vartheta_i - \sum_{j:j \neq i} w_{ij} s_j > 0$ and

the maximal decrease in the energy is attained when $s'_i - s_i$ is the smallest, that is if $s'_i = 0$. Thus, in essence the activation function of the Hopfield model greedily improves of the value based on its immediate neighborhood. This analysis also shows us that the activation rule guarantees that the overall energy does not increase, and therefore disregarding possible plateaus in the energy, the energy is guaranteed to converge at a local minimum.

Bearing in mind that the activation rule of the Hopfield model performs greedy optimization of a single variable based on its immediate environment, we can easily expand it to our framework of general potentials by defining that a variable X_i attains a value based on the following rule:

$$X_i = \arg\min_{x_i} \varphi_i(x_i) + \sum_{\{i,j\} \in E} \psi_{i,j}(x_i, x_j)$$

for energy minimization, and changing the min with max for energy maximization.

1.3.5 Graph Cuts

Graph cuts[4] is an energy minimization via local search algorithm specialized for approximating energy functions represented as sums of binary potentials, like those that we are looking at. However, in addition to this constraint on the structure of the energy function the common variant of graph cuts, which improves the current assignment by what is known as "α-expansion" steps, requires that the the potentials will be metrics, that is that for every i and j and for every three values α, β and γ the following conditions apply: $\psi_{i,j}(\alpha, \beta) = \psi_{i,j}(\beta, \alpha) \geq 0$ (symmetry) , $\psi_{i,j}(\alpha, \beta) = 0 \Leftrightarrow \alpha = \beta$ (positivity) and that $\psi_{i,j}(\alpha, \gamma) \leq \psi_{i,j}(\alpha, \beta) + \psi_{i,j}(\beta, \gamma)$ (triangle inequality). Another variant of graph cuts, which uses a step known as "α-β swap" eliminates the requirement that the potentials will fulfill the triangle inequality at the cost of lesser performance. Given an assingment $\bar{X}$ and a value α, another assignment $\bar{X}'$ is an α-expansion step away from $\bar{X}$ if for every i $X'_i = X_i$ or $X'_i = \alpha$. That is, every variable in the new assignment may either keep its original value or change its value to α. A transition from $\bar{X}$ to $\bar{X}'$ is an α-β swap if for every i, if $X_i \in \{\alpha, \beta\}$ then $X'_i \in \{\alpha, \beta\}$ and otherwise $X'_i = X_i$. That is only variables which currently have a value of α or β may change their value, and they may do so by switching from α to β or vice versa. For both variants of graph cuts the main idea behind the algorithm is to iteratively improve the current assignment by finding the optimal assignment given a value α (or values α and β for α-β swapping) under the constraints defined by the step. As implied by the algorithm's name, this is done by reducing the problem to a graph flow problem and solving exactly. As a result, a main feature of graph cut is that it produces exact solutions if the domains of the variables are of size 2. Additionally, both variants are known to converge at a local minimum after a limited number of iterations and an upper bound on the approximation ratio of graph cuts with α-expansions has been proved[26]. For these reasons, graph cuts is widely used in many computer vision applications in which potentials of this form arise naturally.

Algorithm 1.1 pseudo code of BPLS/BPLS*.
BPLS and BPLS* differ in the restart and the cutset generation procerdures (see text). *is_stagnated* is set to TRUE once no change has been made by both the BP and the local search stages during a given number of iterations.

Input: Graph $G = (V, E)$ annotated with potentials φ_i and $\psi_{i,j}$ and time bound t.
Output: An assignment $\bar{x}$ which achieves the minimum energy found in time t.

```
1  x̄ ← InitializeValues
2  while runtime < t do
3  |  C ← GenerateCutset(G)
4  |  F ← V\C;
   |  // Alternate BP on forest variables and local search on cutset
   |  variables until convergence
5  |  repeat
6  |  |  x̄|_F ← BP_min_sum(G, x̄|_C, F)
7  |  |  x̄|_C ← SubsidiaryLocalSearch(G, x̄, C)
8  |  until no change in x̄;
9  |  if is_stagnated then
10 |  |  x̄ ← InitializeValues
11 |  |  is_stagnated ← FALSE
12 |  end
13 end
```

1.4 BPLS: Belief Propagation-based Local Search

Since performing complete conditioning is not feasible, [24] suggested iteratively conditioning on a different cutset and finding exact optimal solution on the rest variables as a possible scheme for dealing with cycles in the graph. The algorithm can additionally perform regular local search on the cutset variables. However, they did not go further to establish the capabilities of this method, which we call BPLS - Belief Propagation-based Local Search (or STLS - Stochastic Tree-based Local Search). The operation of of BPLS is given in Algorithm 1.1.

1.4.1 Properties of BPLS

1. **Convergence to strong optima.** In every iteration an optimal assignment to the forest variables is generated given the values of the cutset variables using *BP min-sum*. Therefore, the energy of the system can not increase during the process and the resulting algorithm is a local search algorithm finding the optimal solution on all the forest variables in every iteration. For this reason, it is clear that BPLS converges to a local optimum. However, due to the structure of BPLS, in which a step is defined by (possibly) modifying the assignment to all variables in a forest

this "local optimum" is in fact a strong local optimum. Namely, it is conditionally optimal relative to every cycle-cutset.

2. **Complexity.** Since the complexity of BP is $\mathcal{O}(Nk^2)$ where N is the number of variables and k is their domain size, the complexity of each iteration is $\mathcal{O}(nk^2)$ where n is the size of the forest F in addition to the complexity of the subsidiary local search algorithm used on the cutset variables (if used). In practice the algorithm attempts to find the optimal assignment in as many iterations as possible in the given time bound.

1.4.2 Initialization

The two basic variants we consider for variable initialization are either at random or by initializing all variables to a special *undefined* value. In the latter case the cycle-cutest variables holding an *undefined* value are effectively ignored until their value is set to valid one. When the initialization to *undefined* scheme is used, the variables keep their *undefined* value until they are chosen to take part in the forest F and update their value as part of the BP update stage. Additionally, if a subsidiary local search algorithm is used to set the values of the cutset variables, a cutset variable may acquire a valid value if such a value can be defined by the subsidiary local search algorithm (e.g. for Hopfield Model, a cutset variable will set its value once all its neighbors' values have been defined).

1.4.3 Tree Directing

Working only in the context of acyclic graphical models [24] presents a straightforward distributed algorithm for asynchronously directing undirected graphs to produce directed trees from the leaves to a root, assigned as a by-product of the procedure. In the context of general graphs this algorithm becomes a subroutine used by the cutset selection procedure (see Section 1.4.4) in order to "unravel" parts of the graph which are not part of any cycle when removing the current cutset from the graph. For the sake of completeness we present our adaptation of the tree directed algorithm used in BPLS. Initially, all vertices are initialized so that they do not point to any of their neighbors. Given a vertex v not already assigned a parent, if all but one of its neighbors u point to v as their parent, then u is set as v's parent. That is, if $|\{u \in nb(v) : v \neq pa(u)\}| = 1$ then $pa(v)$ is set to be the only element in that set. Then, since this assignment reduced by 1 the number of neighbors of u pointing to it as their parent, the procedure is recursively called on u to check if there remain only a single neighbor of u not pointing at u as its parent. If there v has no neighbors not pointing to it as their parent, v is defined as the root of its tree. Otherwise, that is if $v$has more than 1 neighbor not pointing to it, the procedure terminates without changing the graph. It can be easily seen that given an acyclic graph calling this procedure on every vertex once (in any order) would result in directing all edges from the leaves toward the single root of every connected component.

1.4.4 Cutset Selection

The cutset selection algorithm is based on the algorithm described in [2], where the cutset is gradually built by adding one vertex at a time. In principle, the graph's vertices V are divided to three disjoint groups: The cutset vertices C, i.e. vertices already assigned to the cutset; The forest vertices F, i.e. vertices who are not part of any cycle in $V \backslash C$, and the remainder vertices R. In every iteration a vertex from R which has a more than 2 neighbors in R is randomly chosen and added to the cutset vertices C with probability proportional to its number neighbors in R. Once no variable in R has more than 2 neighbors in R, i.e. once the induced graph on R is a union of disjoint simple cycles, each cycle is opened by randomly picking a variable from each with uniform probability. After each new cutset vertex $v$is chosen the tree directing algorithm of [24] is run on its neighbors in R, in order to check if the new cutset variable's removal allowed moving them to forest vertices F.

Motivated by the ideas presented in [24], in our implementation the probability a variable is chosen is governed not only by its degree, but also by the number of iterations in which it has not changed its value and by the number of iterations it was not a part of the cycle-cutset. several linear combination of the aforementioned parameters were tested, but no major difference was noted.

In order to reduce the cutset's size, a message is passed from the roots down to the leaves after the forest is formed, notifying the variables what is the root of their tree, and essentially defining explicitly the partition of the forest into trees. Then, each cutset variable which is not a neighbor of two variables belonging to the same tree is move from the cutset to the forest variables, as doing so necessarily will not form a cycle. Once all the redundant cutset variables have been removed, the tree forming algorithm is run again (only on the forest variables) in order to form a well directed forest.

In addition, a new variant on this algorithm was tested. In this variant, in order to encourage the formation of highly connected forests, i.e. forests in which the number of trees is (relatively) small and which are mainly composed from a single big tree, the first stage of [2] is split to 2 stages. That is, at first the algorithm tries to add to the cutset vertices (from R) which are **not** already a parent of another vertex as defined by the tree directing algorithm of [24], since these are vertices acting as junctions connecting vertices from F to vertices from R. Thus, adding such a vertex to the cutset will separate its children (which are in F) from the trees to be formed in R, which will ultimately increase the number of connected component. Only when no such vertices remain, vertices acting as parents are added to the cutset in accordance to the original algorithm.

1.5 BPLS for Panorama Stitching

In the problem of panorama stitching several aligned images (presumably taken from the same scene) are required to be merged together so that the seams between the various images will not be visible. To do so, an energy function that expresses the discrepancy

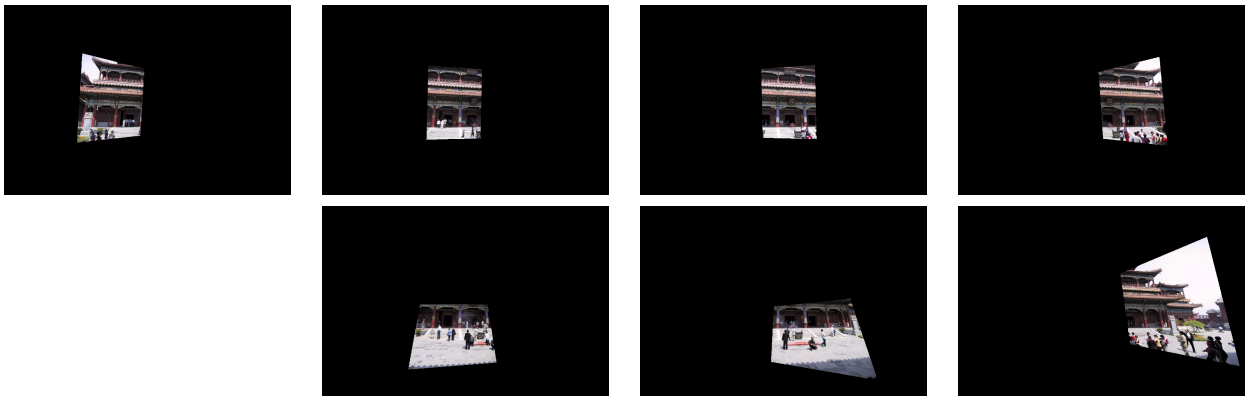


Figure 1.1: Input images for the panorama stitching benchmark.

between each pair of neighboring pixels is generated and **minimized**. Specifically, for every two neighboring pixels p and q a potential $V_{p,q}$ of the following form is defined:

$$V_{p,q}\left(l_p, l_q\right) = \left|S_{l_p}\left(p\right) - S_{l_q}\left(p\right)\right| + \left|S_{l_p}\left(q\right) - S_{l_q}\left(q\right)\right|$$

where l_p is the label assigned to the pixel p, and $S_{l_p}\left(p\right)$ is the color value of if the l_p'th image at the pixel p. Additionally, for every pixel p an unary potential $d_p\left(i\right)$ is defined to be 0 if p is in the field of view of the image i and ∞ otherwise, in order to force that every pixel will be assigned a value from a valid image. Thus the energy function to be minimized is

$$\mathcal{E}\left(\bar{l}\right) = \sum_p d_p\left(l_p\right) + \sum_{p,q} V_{p,q}\left(l_p, l_q\right)$$

As metioned earlier, these kind of problems are generally by solved by graph cuts[4] which is specifically designed to deal with potentials which are generated from a metric.

In order to assess the performance of BPLS compared to specialized algorithms such as graph cuts, we have conducted experiments on a single instance from this domain taken from [25, 1][1], which contains 7 images to be merged presented in Figure 1.1.

Table 1.1 presents a summary of the resulting energies for the BPLS with random initialization and initialization to *undefined* with updating all forest variables in the BP step (BPLS - *basic*) and updating only trees bigger than 10% of the graph (BPLS - *Big Trees*) after several time periods. From the table it is evident that the randomly initialized variant is completely outperformed by the other variants. In addition, the results suggest that updating only big trees during the BP step is preferable to updating all the variables in this domain. However, while this domain was used as a main testbed and motivation for the heuristics devloped for BPLS*, this benchmark was not further investigated, since graph cuts with the expansion step achieves an energy of 151,261 within

[1] http://vision.middlebury.edu/MRF

Table 1.1: Summary of results for the panorama stitching problem. Presented are the resulting energies in thousands after several time periods. BPLS - *Basic* is basic implementation of BPLS. BPLS - *Big Tree* differs from BPLS - Basic in that only trees whose size is bigger than 10% of the graph are updated in the BP step. BPLS - *Random* differ for BPLS - Basic in that the variables are initialized randomly.

Algorithm		1 min	3 sec	20 min
BPLS - Basic	min	553	348	348
	max	709	494	387
BPLS - Big Trees	min	338	286	286
	max	447	382	331
BPLS - Random	min	7519	5066	-
	max			

24 seconds, showing that BPLS cannot compete with such a specialized algorithm on this benchmark.

1.6 Experiments on BPLS

1.6.0.1 Methodology

Several problems sets from the PASCAL2 Probabilistic Inference Challenge (PIC2011)[2] were used as our main benchmarks for evaluating the performance of energy minimization algorithms. Since BPLS was implemented only for binary potentials, we used domains supporting this constraint: Grids, CSP, Protein Folding and Segmentation (see [15] for a summary of the statistics of each of these benchmark sets). The *Grids* domain is composed of graphical models structured as grids or as grids with wrap around (i.e the leftmost variable in every row is connected to the rightmost variable at the same row and the top variable at every column is connected to the bottom variable at the same column). The sizes of the grids vary among the sizes of 20×20, 40×40 and 80×80 variables. The *CSP* problems are restatements of constraint satisfaction problems in the form energy maximization problems. Table 1.2 provides statistics for a subset of the instances from each benchmark . It summarizes the number of variables n, the number of potentials f, the maximum domain size k and the approximated induced width w of the main problems used. These are given in blocks corresponding to the different domains (Grids, CSP, Protein Folding and Segmentation from top to bottom).
Each algorithm or variant was run on each problem instance 10 times for a bounded period of time of of up to 3 minutes and the best assignments achieved after 0.1, 1, 10, 60, 120 and 180 seconds were registered. The Segmentation problems are relatively small and converge faster, the algorithms were run on this domain for only one minute. It should

[2] http://www.cs.huji.ac.il/project/PASCAL/index.php

Table 1.2: Problem sets statistics: The number of variables n, number of potentials f, domain size k and approximated induced width w for the problem instances in the problem sets.

Problem name	n	f	k	w
Grids				
grid20x20	400	1161	2	24
grid20x20.XXX.wrap	400	1201	2	44
grid40x40	1600	4721	2	52
grid40x40.XXX.wrap	1600	4801	2	95
grid80x80	6400	19041	2	106
grid80x80.XXX.wrap	6400	19201	2	196
CSP				
CELAR6-SUBX	14 - 22	~300	44	10
DSJC125.1.4	125	737	4	65
GEOM30a_X	30	82	X	6
driverlog01ac	71	619	4	9
le450_5a_X	400	5715	X	315
myciel5g_X	47	237	X	21
queen5_5_X	25	161	X	18

Problem name	n	f	k	w
Protein Folding				
pdb1d2e	1328	5220	81	22
pdb1iqc	1040	4042	81	26
pdb1kgn	1060	4715	81	38
pdb1kwh	424	1881	81	27
pdb1m3y	1364	5037	81	29
pdb1qks	926	3712	81	36
Segmentation				
2_2_s.21	227	845	21	16
2_2_s.binary	227	845	2	16
3_16_s.21	229	852	21	18
3_16_s.binary	229	852	2	18

be noted that if by the time of the measurement the assignment was invalid, i.e. the variables were initialized to *undefined* and not all of them acquired a valid value, then the assignment was completed randomly only for the purpose of the measurement, without affecting the operation of the algorithm.

1.6.0.2 Variants

We have experimented with four different variants of BPLS: In the *Basic* variant the variables are initialized to the *undefined* value, all the the forest variables updated during the BP step, and the cutset variables are updated using the Hopfield model activation function (once all their neighbors have acquired a valid value) as the local search algorithm mentioned in line 7 of Algorithm 1.1. The *Random* variant differs from *Basic* in that the variables are initialized randomly. the *No Hopfield* variant differs from *Basic* in that it does not perform an additional local search step, i.e. the assignment is improved solely by the BP step. The final variant, *Big Trees,* differs from *Basic* in that in the BP step only trees bigger than 10% of the entire graph are updated.

Table 1.3: Experimental results on different variants of BPLS. The values presented refer to the average and maximal results over 10 runs obtained after 1 minute for the Segmentation set and 3 minutes for all other sets. *Random* is BPLS with random initialization, *No Hopfield* is BPLS with only the BP stage, *Big Trees* is BPLS with where only trees of size bigger than 0.1 of the graph size are updated in the BP stage (and cutset nodes updated using Hopfield), and *Basic* is BPLS with all trees updated in the BP stage. *Best* is the number of the instances for whom the algorithm achieved the best result (and second best in parenthesis). *Ratio* is the average ratio of the result obtained by the algorithm to that of BPLS with small trees update. For the basic variant of BPLS the average ratio of the result to the best overall result is presented.

Set (# instances)		Random		No Hopfield		Big Trees		Basic	
		Best	Ratio	Best	Ratio	Best	Ratio	Best	% of best
Grids (21)	mean	0 (0)	0.84	2 (7)	0.99	**10 (7)**	**1**	9 (7)	**95%**
	max	0 (1)	0.90	4 (4)	0.98	5 (10)	0.99	**12 (6)**	**99%**
CSP (29)	mean	**26 (2)**	**1.01**	17 (6)	1	12 (6)	0.98	14 (7)	85%
	max	**28 (1)**	**1.01**	20 (6)	1	19 (6)	1	22 (4)	86%
Protein. (9)	mean	3 (0)	**1**	2 (4)	**1**	0 (0)	0.98	**4 (5)**	**100%**
	max	6 (2)	**1**	8 (1)	**1**	1 (4)	0.99	**8 (1)**	**100%**
SGM. (90)	mean	**65 (10)**	**1.03**	56 (20)	1	0 (49)	0.85	58 (16)	94%
	max	**90 (0)**	**1.05**	70 (17)	1	53 (25)	0.97	69 (14)	97%

1.6.0.3 Results

As mentioned, each variant was run on every problem instance 10 times independently. For each problem instance the mean of the results and best result over all runs were calculated. It should be noted that the mean characterizes the behavior of a single run and the best result characterizes the combined behavior of all 10 run. Since the resulting energies are arbitrary and meaningless out of context, for each instance all the results over all times and all algorithms in the comparison were linearly normalized to the interval $[0, 1]$, mapping the worst result to 0 and the best to 1. A summary of the results at termination is displayed in Table 1.3. For every problem set the column *Best* indicates the number of problems on which each algorithm achieved the best results. In order to facilitate easy comparison of the different variants to the basic BPLS, the ratio of the (normalized) result obtained by each algorithm to the result obtained by *Basic* was calculated. The average ratio over all problems in a single problem set is presented in the column *Ratio*. In order to describe the overall behavior of the algorithms, the column "*% of best*" gives the average ratio of the result obtained by *Basic* to the best result obtained by any of the compared variants. The best results for each of the 2 metrics is written in bold font for every measurement.

As can be seen in the table, using Hopfield on the cutset variables does not have much impact on the resulting energies as the results of *No Hopfield* are very similar to those of

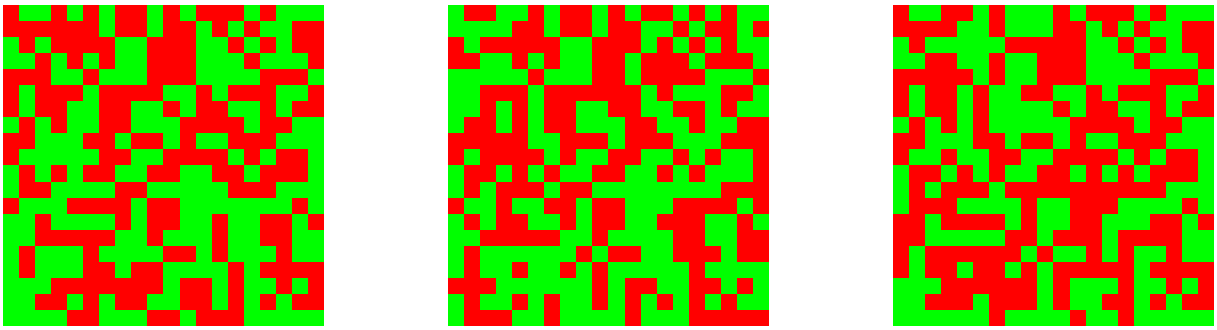


Figure 1.2: Several solutions obtained by BPLS for the problem grid20x20.f10 from the *Grids* benchmark. Each variable is denoted by a colored square. A value of 0 is denoted in red and a value of 1 is denoted in green.

Basic. Similarly, updating only big trees does not significantly affects the performance of BPLS on the Grids and CSP problem sets, but does impair its performance on the Protein Folding and Segmentation benchmarks. The randomly initialized version exhibits the best performance on the three less structured benchmarks with lower induced width, i.e. CSP, Protein Folding and Segmentation, but fails considerably on the Grids benchmarks. For this reason, and since the randomly initialized version does not significantly outperform the *Basic* version on the other benchmarks, it appears that the *Basic* version displays the best overall performance among these algorithms..

1.7 BPLS*

A more complex version of BPLS, named BPLS*, attempts to use additional heuristics for the cutset selection and assignment perturbation, that is the so-called "restarts". These heuristics are based on the values of the potentials and on information accumulated by the algorithm about assignments on which it has previously stagnated.

1.7.1 Correlation Estimation Based Value Perturbation

Closer inspection of the several approximated solutions to some of the Grids problems (see Figure 1.2) indicates that the variables in certain regions of the graph are strongly correlated to one another, i.e. given the value of one variable the values of neighboring variables can be deduced. This implies that the variance in the energy function around local optima is mainly governed by the interaction along the seams between these correlated regions, while the regions themselves operate in tandem. This supposedly occurs due to strong interactions inside the correlated regions and weaker interactions between the regions, much like the seams in panorama stitching. The fact that the graph can be partitioned to regions operating in unison implies that the problem can be naturally reduced by focusing on the interactions between the regions, while maintaining the inner

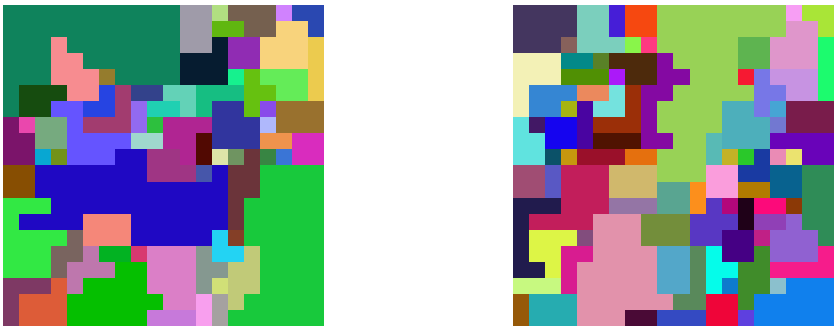


Figure 1.3: Estimated correlation maps. Each uniformly colored region represents an estimated correlated group in the problem grid20x20.f10 after 180 seconds. Note that smaller regions in each image are obtained as refinements of larger regions in the other, but the overall structure is consistent between the two partitions.

relations within each region. For this reason attempts to fathom the correlated regions on the fly were made. Therefore, every time BPLS* stagnates on a local optimum the values of each pair of neighboring variables were recorded. A node u was defined as correlated to a node v if for every value x_v of v, there exists a value x_u of u, such that $X_u = x_u\,(x_v)$ in more than 90% of the cases where $X_v = x_v$. The relation is then applied transitively and symmetrically to assess entire correlated regions (see Figure 1.3).

The correlation estimations are aimed to be utilized during restarts such that the values of all variables in a group would be perturbed in concert, thus suitably keeping the inner relations which were already estimated. Specifically, whenever BPLS* is declared as stagnant and a restart is required, the values of all the variables in some of the estimated correlation groups were changed while keeping the values of all the variables in every group appropriately matched to all other variables in the group. However, preliminary experimentation with this approach did not prove to be considerably fruitful, and a different, simpler mechanism was suggest and is described in the next section.

1.7.2 Experience based value perturbation

Due to the underwhelming results of the previous approach a simpler mechanism was suggested, in which instead of measuring the frequency in which each pair of values appears for every two neighboring variables, only the appearance of each (single) value of every variable is counted. That is, a counter is set for every possible value x_v of every variable v, which counts the number of times that v assumes the value x in an assignment on which the BPLS* stagnates. Thus, instead of storing $\mathcal{O}\,(N^2k^2)$ entries, where N is the number of variables and k is a bound on the size of the domain, only $\mathcal{O}\,(nk)$ entries are stored.

In general, the restart procedure (see Algorithm 1.2) alternates between restarting from a completely random assignment and setting only a part of the assignment of the

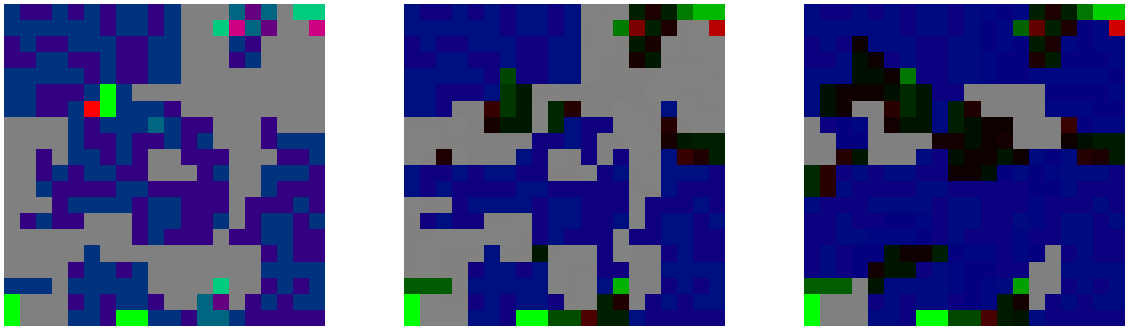
Algorithm 1.2 pseudo code of the BPLS* restart procedure. x_v is the current value of variable v. counter is initially set to 0 when BPLS* is run and it retains its value between consecutive calls to restart during the same run.

```
// Register the values of the variables at stagnation.
foreach v ∈ V do
    value_counter [v,x_v] ← value_counter [v,x_v]+1
end

/* In case the current assignment seems promising, only prune unrealiable
values and continue.  Otherwise - restart normally.                    */
if new_best_assignment then
    UndefineUncertainNodes(C)
else
    if restart_counter modulo 2 = 1 then
        setRandomAssignment(C)
    else
        UndefineUncertainNodes(C)
        if restart_counter modulo 6 = 2 then
            FortifyCertainNodes(C)
        end
    end
    restart_counter ← restart_counter +1
end
new_best_assignment ← false
```

undefined value and attempting to better reassign them based on the values of the rest of the variables.

By alternating between the two restart methods, BPLS* attempts to balance exploring new regions in the assignment space and trying further improve the current assignment, which has already been proved as fair, being an assignment on which BPLS* stagnated. The variables assigned to *undefined* in the latter scheme are those whose values in stagnation are most evenly distributed among their possible values, i.e. the variables for which a prominent value has appeared to the least extent. This approach is based on the assumption that variables with dominant values will hold that value also in a global optimum, and that that global optimum can be deduced from their values. In addition, occasionally, a certain number of variables for which a dominant "value in stagnation" appears is set to that dominant value (if not already holding it). Thus, uncertain values are pruned and certain values are fortified, allowing the algorithm to try to improve from a hopefully preferable assignment (see Figure 1.4).



(a) 1 second (b) 60 second (c) 180 second

Figure 1.4: Value counts for the problem grid20x20.f10 at several time bounds. Each variable in the 20-by-20 grid is represented by a square. The intensity of the square's color indicates its dominance in stagnation. Variables appearing with a gray tone will be set to *undefined* upon stagnation. Variables appearing with a blue tone will keep their current values. The rest of the variables present a dominant value and will be set to it.

1.7.3 Experience based cutset selection

The value counters are not used only for restarting, but play a role in the cutset selection process as well. For every variable X_v a prominence index, which estimates the the level of confidence in its assignment, is defined as follows: let n_i for $1 \leq i \leq k$ be the number of time stagnation was declared when $X_v = i$ and let n be the total number of time X_v had a valid number in stagnation (i.e. not *undefined*). Define $m = \arg\max_{1 \leq i \leq k} n_i$ to be a value which X_v holds most often at stagnation and $r = \frac{n_m}{n}$ the maximal frequency. Then the prominence index is

$$I_v = \frac{rk - 1}{k - 1}$$

Where k is the domain size of X_v. Clearly, I_v is defined by a linear function which is 0 when X_v attains all possible values with equal frequency and $I_v = 1$ if X_v attains only a single value.

In order to improve the assessment of variables for which no significant value appears, a variable v may be added to the cutset with probability proportional to I_v, in addition to selecting a variable with probability proportional to its number of neighbors in R as discussed in 1.4.4. That is, variables whose value in an optimal solution is predicted with a high estimated confidence are more likely to be added to the cutset than those whose value is predicted with a low certainty. This heuristic is based on the conception that the BP is the main engine for finding a good assignment for the variables, and therefore variables whose prediction is uncertain are preferred to be handled as part of the BP step.

1.7.4 Potential based cutset selection

When reviewing the cutset selection procedure, it is apparent that the energy minimization problem at hand affects the selected cutset only in a limited way by the structure of the problem, while the values of the potentials themselves do not play a role. In an attempt to prevent this and out of an assumption that BP produces more robust results, we aim to handle variables with a significant effect on the energy function within the frame of BP as much as possible. Therefore, for every variable an estimation of the extent of its potential effect on the function is generated. When picking variables to the cutset, a variable with a higher effect is less likely to be selected for the cutset. In particular, the potential effect of a variable v is estimated as the maximum change to the function brought by changing v's value to any other value, while all the other variables retain their current values. Once all the estimates have been calculated they are normalized such that the maximum estimation is set to 1. In BPLS*, a variable v with potential estimate pe is selected to the cutset with probability proportional to $1 - pe$, independently from the probability described in Section 1.4.4.

1.8 Experiments on BPLS*

Due to the minor impact of removing the Hopfield step in BPLS and due to the combined restart scheme of BPLS*, which prevents initializing the variables randomly, we have experimented with two variants of BPLS*: BPLS* - *basic* in which all forest variables are updated during the BP stage, and BPLS - *Big Trees* in which only trees of size bigger than 0.1 of the graph size are updated. These variant are derived from their BPLS counterparts by applying the aforementioned heuristics. Table 1.4 summarizes the results of the experiments conducted according to the procedure described in section 1.6.0.1.

It can be seen in the Table 1.4 that in every benchmark the BPLS* variants outperform their BPLS counterparts both in the number of instances for which they achieve the best results among the compared algorithms and in the overall quality of the results (relatively to those of BPLS - Random). This testifies on the contribution of the heuristics to further improve the performance of BPLS. Furthermore, we can see that BPLS* - basic outperforms all other variant on all benchmarks on both measures excluding the *Best* measure for maximal value in the Segmentation benchmark, making it the definite leader among these variants.

1.9 Comparison with GLS$^+$

In order to receive a more educated evaluation of the performance of the BPLS variants, we have compared it to GLS$^+$ which is the fruit of extensive development and research and holds the status the state-of-the-art for nearly a decade. For this purpose, GLS$^+$ was run

Table 1.4: Evaluation of BPLS*. The values presented refer to the average and maximal results over 10 runs obtained after 1 minute for the Segmentation set and 3 minutes for all other sets. Variants denoted as *Basic* are those in which all the forest variables are updated as opposed to those denoted by *Big Trees*, in which only trees of size bigger than 0.1 of the graph size are updated in the BP stage. BPLS - *Random* is BPLS with random initialization. *Best* is the number of the instances for whom the algorithm achieved the best result (and second best in parenthesis). *Ratio* is the average ratio of the result obtained by the algorithm to that of BPLS - Random. For the BPLS - *Random* the average ratio of the result to the best overall result is presented.

Set		BPLS* - Basic		BPLS* - Big Trees		BPLS - Basic		BPLS - Big Trees		BPLS - Random	
(# instances)		Best	Ratio	Best	Ratio	Best	Ratio	Best	Ratio	Best	% of best
Grids	mean	**16 (2)**	**1.25**	2 (16)	**1.25**	2 (2)	1.23	1 (1)	1.23	0 (0)	78%
(21)	max	**10 (9)**	**1.15**	7 (8)	1.13	3 (2)	1.12	1 (2)	1.11	0 (0)	87%
CSP	mean	**20 (5)**	**1**	14 (8)	0.99	14 (0)	0.99	12 (1)	0.98	19 (6)	**85%**
(29)	max	**24 (2)**	**1**	**24 (0)**	**1**	21 (2)	0.99	19 (3)	0.99	**24 (4)**	**86%**
Protein.	mean	**6 (3)**	**1**	0 (0)	0.99	2 (4)	**1**	0 (0)	0.99	1 (2)	**100%**
(9)	max	**9 (0)**	**1**	2 (1)	0.99	8 (1)	**1**	1 (3)	0.99	6 (3)	**100%**
SGM.	mean	**63 (16)**	**1.05**	1 (32)	0.95	57 (6)	1.03	0 (27)	0.92	58 (14)	95%
(90)	max	82 (5)	**1**	59 (19)	0.98	68 (10)	0.97	53 (19)	0.95	**84 (5)**	**100%**

on all benchmarks as described in section 1.6.0.1 using both a random initial assignment and a more sophisticated assignment generated using Mini-Buckets. In addition, we experimented with a simple hybrid of BPLS and GLS$^+$, in which GLS$^+$ and BPLS - basic are run alternately until each declares stagnation and then the other algorithm attempts to further improve the best assignment currently found. In this case, both algorithms can be seen both as initialization schemes for the other one and as mechanisms for extracting each other from local optima.

Table 1.5 summarizes the results of the comparison to GLS$^+$.

We can see in Table 1.5 that all algorithms performs similarly on the Protein Folding benchmark. In the Segmentation benchmark, while the pure GLS$^+$ variants outperform the rest of the algorithms in the *Best* measure, we can see that the overall performance of the GLS$^+$ algorithms as portrayed by the ratios does significantly exceed that of the hybrid and BPLS* and is equal to that of randomly initialized BPLS. Comparably, the hybrid clearly presents the best performance in the Grids benchmark in regard to the *Best* measure, but the hybrid and BPLS* only sightly outperform GLS$^+$with Mini-Buckets while significantly outperforming the randomly initialized GLS$^+$. Finally, while the algorithms perform similarly on the *Best* measure in the CSP benchmark, we can see that the hybrid and BPLS* significantly outperform the rest of the algorithm in the *Ratio* measure, implying that while the algorithms operate alike on most problems, there are some problems on which these two algorithms substantially outperform the rest. This can be explicitly

Table 1.5: Comparison with GLS$^+$. The values presented refer to the average and maximal results over 10 runs obtained after 1 minute for the Segmentation set and 3 minutes for all other sets. *GLS*$^+$ is GLS$^+$ initialized using Mini-Buckets and *GLS*$^+$ - *Random* is GLS$^+$ initialized randomly. *Hybrid* is the hybrid of BPLS and GLS$^+$. *Best* is the number of the instances for whom the algorithm achieved the best result (and second best in parenthesis). *Ratio* is the average ratio of the result obtained by the algorithm to that of BPLS - Random. For the BPLS - *Random* the average ratio of the result to the best overall result is presented.

Set		GLS$^+$		GLS$^+$ - Random		Hybrid		BPLS* - Basic		BPLS - Random	
(# instances)		Best	Ratio	Best	Ratio	Best	Ratio	Best	Ratio	Best	% of best
Grids	mean	0 (14)	1	0 (0)	0.74	**19 (1)**	**1.02**	2 (6)	1.01	0 (0)	83%
(21)	max	0 (6)	1	0 (0)	0.86	**14 (6)**	1.03	7 (8)	**1.05**	0 (1)	91%
CSP	mean	**19 (4)**	1	18 (5)	1	13 (6)	1.14	17 (2)	**1.16**	15 (7)	85%
(29)	max	17 (8)	1	16 (9)	1	17 (6)	1.15	**21 (2)**	**1.17**	**21 (3)**	86%
Protein.	mean	**6 (1)**	**1**	5 (2)	**1**	2 (1)	**1**	2 (3)	**1**	0 (2)	**100%**
(9)	max	4 (3)	**1**	4 (2)	**1**	**5 (2)**	**1**	7 (0)	**1**	**5 (0)**	**100%**
SGM.	mean	**55 (34)**	**1**	51 (36)	**1**	44 (36)	0.98	36 (21)	0.95	27 (21)	**92%**
(90)	max	49 (37)	**1**	**51 (39)**	**1**	49 (41)	0.99	46 (34)	0.98	45 (33)	**98%**

seen in figure A.4 in the appendix, where it is apparent that the GLS$^+$ fail considerably compare to the BPLS algorithms on the CLEAR6 problems. All in all it should be noted that the pure GLS$^+$ variants never outperform both BPLS variants simultaneously in the *Ratio* measure, meaning that the overall performance of BPLS is at least comparable to that GLS$^+$on these benchmarks, and it is even superior to performance GLS$^+$as exhibited by the CSP benchmark.

In addition, as shown by the detailed figures in the appendix, due to the complex initialization scheme of the standard version of GLS$^+$, it requires considerably more time in order to produce even an initial result, while the simple initialization schemes of BPLS allow it to produce results very quickly. In many cases these results are reasonably close to the end result found at the experiment's termination. This is most apparent in the 80×80 grids, where it takes GLS$^+$ more than 10 seconds to produce the first result for many of the problems, while the BPLS variants produce results which are about 90% of the maximal results in about 1 second (and even the results after merely 0.1 second are fairly reasonable).

Finally, it is should be noted that the performance the hybrid algorithm is similar to that of the best of GLS$^+$ and BPLS. That is, by joining both algorithms one can gain the benefits of both, meaning that they indeed operate as mechanism for extracting each-other from local optima and that the time spent in each is well spent.

1.10 Discussion and Future Work

In this chapter we presented BPLS, a stochastic local search algorithm for energy minimization, which combines the notion of cycle-cutset with the well known Belief Propagation algorithm to achieve an approximate optimum of a sum of unary and binary potentials. This is done by the previously **unimplemented** concept of traversal from one cutset to another and updating the induced forest, thus creating a local search algorithm, whose update phase spans over many variables (the forest variables). We have presented experiments comparing the performance of different variant of BPLS and evaluating different aspects of the algorithm. The panorama stitching benchmark suggests that in extremely problems it is preferable to update only sizable trees. However, BPLS is clearly outperform in this domain by the specialized Graph Cuts. This superiority of updating only big trees was not observed in other benchmarks, leaving the basic implementation initialized either randomly or using the special *undefined* value as the leading variants of the simple BPLS.

Furthermore, We suggested several heuristics aimed at learning the structure of the input problem, reusing information collected in previous iterations and using the problem structure in a more educated manner in the cutset selection and the value perturbation steps. We have presented empiric results indicating the contribution of these heuristics in structured problems such as grids while achieving results at least comparable to the other BPLS variants on other problem sets. Thus, making BPLS* - Basic the definite leader among all BPLS variants.

Finally, we have presented experiments indicating that this algorithm beats GLS$^+$, the consistent state-of-the-art, in the Grids and the CSP domains, especially in shorter times, while using only a basic form of initialization. Additionally, we have shown that in other case the strongest BPLS variants are at most only slightly outperformed by GLS$^+$ on energy function described as a sum of unary and binary potentials. Moreover, we have seen that the two algorithms can be effectively combined to produce an algorithm that improves GLS$^+$ and BPLS on many problems, both in time and in quality.

In future work, BPLS should be extended to handle potentials of higher arity than 2. One possible way to achieve this is to find cycle-cutset in the primal graphs resulting from these problems, in which 2 variables are neighbors if they appear in the scope of a single potential. Another possibility is applying the BPLS paradigm directly to the hypergraph induced by these problems. Another way in which this work can be expanded is to convert the notion improving the current assignment by finding an exact solution to the complementary of a forest, that is the complementary of a sub-graph of treewidth 1, to finding an exact solution to a sub-graph of arbitrary treewidth. This will hopefully create an adjustable tradeoff between the complexity of the improvement steps and the quality of the results. On the technical side, BPLS can be further improved by implementing it using parallelism (perhaps even on a GPU). We expect that this would be a fairly straightforward modification, since the original algorithm suggested by [24] was designed to operate in a distributed system.

Additionally, the algorithm should be further investigated in order to understand more fully the parameters governing its behavior, in an attempt to stabilize the results produced by the algorithm on the better side. Importantly, BPLS yields strong local optima, and therefore, in the limit it is as good as max-sum/min-sum belief propagation in quality, while it can be more effective computationally (i.e., guaranteed convergence). Comparing with specific loopy belief propagation scheme is left for future work as well.

Chapter 2

The Minimum Cycle-Cutset Problem in Grids

2.1 Introduction

Definition 7 (The $n \times m$ Grid Graph)**.** Let m, n be positive integers. T*he $n \times m$ grid graph* $M_{n,m}$ is an undirected graph whose vertex set is $V(M_{n,m}) = \{v_{i,j} : 1 \leq i \leq n,\ 1 \leq j \leq m\}$ and the edge set $E(M_{n,m})$ is defined by

$$\begin{aligned} E(M_{n,m}) = &\{(v_{i,j}, v_{i+1,j}) : 1 \leq i < n,\ 1 \leq j \leq m\} \\ &\cup \{(v_{i,j}, v_{i,j+1}) : 1 \leq i \leq n,\ 1 \leq j < m\} \end{aligned}$$

Definition 8 (The Minimum Cycle-Cutset Problem)**.** Let $G = (V, E)$ be an undirected graph. *The minimum cycle-cutset optimization problem* is finding the minimum size of a subset of vertices $C \subseteq V$ such that C is a cycle-cutset of G.

As mentioned earlier, the minimum cycle-cutset problem was proven to be $\mathcal{NP}$-complete for general graphs[12]. Nevertheless, it has been extensively studied, due to its importance in a wide variety of applications, including distributed computing and artificial intelligence in the context of Bayesian inference and constraint satisfaction. Some of the findings include polynomial algorithms finding a minimum cycle-cutset for some specific graph classes and lower and upper-bounds on the size of the minimum cycle-cutset of others. In particular, [16] has previously presented lower and upper bounds on the size of the minimum cycle-cutset of grids. They have shown that the minimum cycle-cutset of the $n \times m$ grid $M_{n,m}$ is of size at least

$$\frac{(m-1)(n-1)+1}{3}$$

and at most

$$\frac{mn}{3} + \frac{m+n}{6} + o(m,n)$$

The upper bound of [16] was later significantly improved by [17], who have shown an upper bound matching the lower bound of [16] in many cases and differing from it by at most 2 in other cases excluding when $m = 5$ and $n \geq 5$. In the following we will show that in grids there always a minimum cycle-cutset, whose complement induces a single tree. This will allow us to improve to the lower bound of [16] by one in some cases, thus closing the gap with the upper bound of [17].

2.2 Preliminaries and Convention

Remark 9. In this chapter we follow a convention by which in all the grid diagrams cutset vertices are denoted by dotted circles, forest variables, i.e. vertices not it the cutset, are denoted by a solid circles, and vertices whose status is unknown or irrelevant are denoted by dashed circles. Accordingly, edges *not* in the graph induced by the complement of the cutset are drawn with a dotted line, edges in the induced graph are drawn with a solid line and edges whose status is unknown or irrelevant are drawn with a dashed line.

Definition 10 (Partition to trees)**.** Let $G = (V, E)$ be an undirected graph, and $C \subseteq V$ a cycle-cutset in G, that is the graph F induced by $V' = V \backslash C$ on G is a forest. We define *the partition T of F to trees* as

$$T = \{t = (V_t, E_t) : \text{ t is a connected component of F}\}$$

i.e. T is a set of (connected) trees, and V' can be written as $V' = \biguplus_{t \in T} V_t$.

Definition 11 (Tree-degree)**.** Let $c \in C$ be a vertex in a cycle-cutset C of G and $t \in T$ a tree induced by C, and denote by $nb(c)$ the neighbors of c in the graph induced by $V' \cup \{c\}$ on G. We define *the tree-degree of c in t under C* to be d, if $|nb(c) \cap V_t| = d$ and for every other tree $t \neq t' \in T$ it holds that $|N(c) \cap V_{t'}| \leq 1$. In general, we say that the tree-degree of c is d if the condition above holds for some t, and that the tree-degree is undefined otherwise. If $d \geq 2$ we call $nb(c) \cap V_t$ the *in-tree neighbors of c.*

Definition 12 (Equivalent cutset vertices)**.** Let $c \in C$ be a cutset vertex and let $c' \in V$ be a vertex of the graph G. We say that c' is *equivalent to c (under C),* if $C \backslash \{c\}$ is not a cutset, but $(C \backslash \{c\}) \cup \{c'\}$ is a cutset.

Definition 13 (Induced degree)**.** Let $c \in C$ be a cutset vertex. The *induced degree of c under C* is the degree of c in the graph induced by $V' \cup \{c\}$.

It should be noted that for every vertex $c \in C$ the induced degree of c is greater or equal to its tree-degree.

It can be shown that the following observations hold:

Lemma 14. ***a.*** *If c is cutset vertex of tree-degree 2, then every vertex along the path between its two in-tree neighbors is of equivalent to c.*

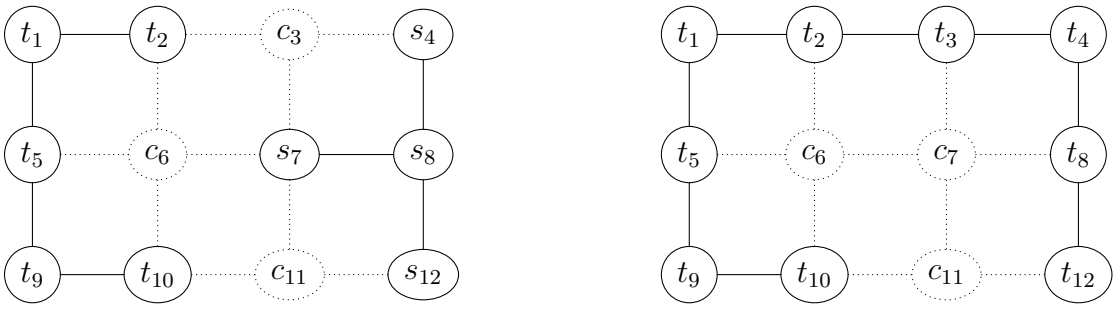
b. *If c is a cutset vertex of tree-degree 3, then there exists a unique vertex c' which is equivalent to c.*

c. *A cutset induces a single tree iff the tree-degree of every cutset vertex is equal to its induced degree.*

Definition 15 (Boundary of a tree)**.** Let $t \in T$ be a tree, we define *the boundary of t* to be all the cutset vertices touching t and some other tree, and denote it B(t), i.e

$$B(t) = \{c \in C : nb(c) \cap V_t \neq \emptyset,\ \exists t \neq t' \in T,\ s.t.\ N(c) \cap V_{t'} \neq \emptyset\}$$

It should be noted that if $c \in B(t)$, then necessarily its tree-degree is strictly smaller than its induced degree.



(a) Original (disconnected) (b) Result (connected)

Figure 2.1: Example of replacement of a cutset vertex of tree-degree 2.

2.3 Connectivity of the induced graphs in grids

Theorem 16. *Let G be a grid graph and let C be a cutset such that the induced forest F is disconnected, i.e. $|T| \geq 2$, then there exists a a series of replacement moves, such that the resulting cutset C' has no more elements than C, and the forest induced by C' contains a single tree, i.e. $|C'| \leq |C|$ and the graph induced by $V \backslash C'$ is connected.*

Corollary 17. *In particular, it follow from Theorem 16 that there exists a minimal cutset whose complement induces a single tree.*

We will prove this theorem using the following Lemmas 18 and 20.

Lemma 18. *Let G be a grid graph and let C be a cutset of G, that the induces a disconnected forest F, i.e. $|T| \geq 2$. Let $t \in T$ be a connected component of F. If there exists a vertex $c \in B(t)$ in the boundary of t with a tree-degree of 2, then c can be replaced with a vertex c', that has a degree of 2 in the graph induced by $V' \cup \{c, c'\}$, thus reducing the number of connected components in the induced forest.*

Example 19. Figure 2.1a presents a 3×4 grid with a cutset $\{c_3, c_6, c_{11}\}$ of size 3 which induces 2 trees - t and s. Under the presented cutset both c_3 and c_{11} have a tree-degree of 2 (in s), as c_3 touches both s_4 and s_7 and c_{11} touches both s_{12} and s_7. However, they both have an induced degree of 3: c_3 since it touches t_2 as well and c_{11} since it touches t_{10}. We see in figure 2.1b that for example c_3 can be replaced with c_7, which has both a tree-degree of 2 as it touches t_3 and t_8 and an induced degree of 2, since it doesn't touch any other forest vertices. In addition, we see that as a result the number of connected component in the induced forest is reduced by 1 from 2 and thus the resulting forest is only a single connected tree.

Proof of Lemma 18. As observed before, c can be replaced with any of the vertices along the path from its two in-tree neighbors. If there exists such a vertex c' with induced-degree 2, then replacing c with c' would reduce the number of connected components in

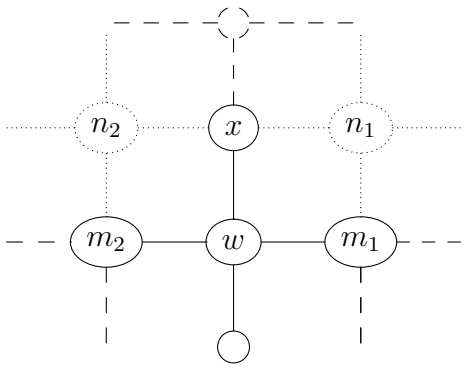


Figure 2.2: Neighborhood of a vertex of degree 4.

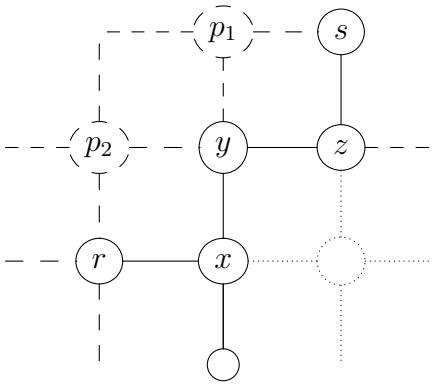


Figure 2.3: Neighborhood of a bend in a path.

the induced graph. To be exact, if the induced-degree of c is p, then the number of connected components would decrease by $p-2$.

We will show that there exists such a vertex c' equivalent to c of induced degree 2. Let u, v be the two in-tree neighbors of c, and assume to contrary that the degrees of all vertices along the path from u to v in the graph induced by $V' \cup \{c\}$ are equal or greater than 3.

Assume that there exists a vertex w along the path of induced degree 4. (refer to Figure 2.2, showing only the neighborhood of w, not necessarily including neither u nor v). One promptly notes that it follows that the following vertex along the path, denoted by x (or the previous one, in case $w = v$) must be of degree 2, in contradiction to the negated assumption. Otherwise, x is of degree at least 3 and either n_1 or n_2 must be in V'. Assume w.l.o.g. that $n_1 \in V'$, then the graph induced by V' contains the cycle w, x, n_1, m_1, in contradiction to the assumption that C is a cutset.

Therefore, assume that all vertices along the path have an degree of 3 in the graph induced by $V' \cup \{c\}$. One notes that the path must not bend, i.e. all the path's vertices must lie on a straight line (in contradiction to the assumption that they form a path from u to v). Assume to contrary that the path bends at vertex y, namely assume w.l.o.g. that the previous vertex x lies below y and that the following vertex z lies to right of y (see Figure 2.3). It follows that x's additional neighbor r (not on the path) must lie to its left (or it will form a cycle r, x, y, z), and similarly z's additional neighbor s must lie above it. therefore, the introduction of y's additional neighbor at each of the possible positions p_1 and p_2 would result in a cycle either with x and r, or with z and s - a contradiction.

It follows that if the boundary of t contains a vertex c of tree-degree of 2, then c can be replaced by an equivalent vertex while reducing the number of connected components in the induced graph. □

Lemma 20. *Let G be a grid graph and let C be a cutset of G, that induces a disconnected*

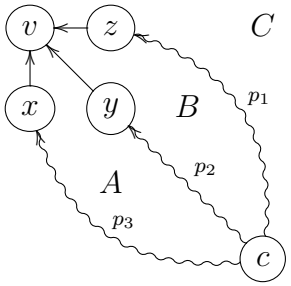


Figure 2.4: Topology of the induced graph in the neighborhood of a vertex equivalent to a cutset vertex of tree-degree 3 in a planner graph.

forest F, i.e. $|T| \geq 2$. If there does not exist a tree $t \in T$ and a vertex $c \in B(t)$ with tree-degree (defined and) equal or less than 2, then:

a. *for every tree $t \in T$, there exist at least 2 vertices in the boundary of t of tree-degree (excatly) 3.*

b. *There exists a series of replacement moves, such that the forest induced by the final cutset C' is composed of less connected components than the original forest.*

Note that the conditions of Lemma 20 are the complementary of the conditions of Lemma 18 (ignoring cutset vertices of tree-degree less than 2, which can be trivially removed).
To prove this lemma we will first show that a vertex in a grid can be equivalent to at most 2 cutset vertices, and that in which case the topology in the neighborhood of the vertex must be of a specific form. To do so, we first set to prove the following general claim.

Lemma 21. *Let $G = (V, E)$ be a planar graph, $v \in V'$ a vertex equivalent to cutset vertices $c_1, \ldots, c_k \in C$ of tree-degree 3, and let d be the degree of v in the graph induced by $V' \cup \{c_i\}_{i=1}^{k}$, then $d \geq k + 2$.*

Proof of Lemma 21. Let $c \in C$ be a cutset vertex of tree-degree 3 and let $v \in V'$ be its (unique) equivalent vertex. Consider Figure 2.4 and note that the three paths p_1, p_2, p_3 from c to v in the graph induced by $V' \cup \{c\}$ partition the plane to three parts, denoted as A, B and C in Figure 2.4(where a solid lines denote a single edge, and a squiggly lines denote a path of arbitrary length):

Assume that there exists another cutset vertex $c' \in C$ of tree-degree 3, such that its equivalent vertex is v as well, and assume w.l.o.g that v lies in section A. In addition, assume to the contrary of the lemma's claim that $d < 4$ (for the d defined in the lemma statement). Then, x, y and z are the only neighbors of v in the graph induced by $V' \cup \{c, c'\}$. Since v is equivalent to c', there must be three paths from c' to v in the graph H induced by $V' \cup c'$. We note that any two of these three paths intersect only at c' and v, as otherwise, there is a path between two in-tree neighbors of c' which does not pass through v, in contradiction to the assumption that v is equivalent to c'.

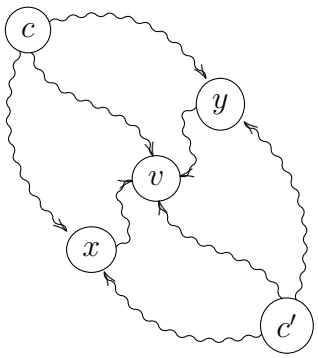


Figure 2.5: Topology of the induced graph in the neighborhood of a vertex equivalent to 2 vertices of tree-degree of 3.

As x, y and z are the only neighbors of v it follows that there must be a path from c' to v in H passing through z, but since G is a planar graph, every path from c' to z in H not passing through v first must intersect with either p_2 or p_3. Assume w.l.o.g that the path from c' to v through z intersects with p_2 at vertex t, then there exist two paths from t to v - one passing through z and the other passing through y, thus forming a cycle in the graph induced by V', in contradiction to the assumption that it is a forest. Therefore, there is no path from c' to z in H not passing through v. As there must be three paths from c' to v, it follows that v must have (at least) one other neighbor in H, through which a third path from c' to v must pass in contradiction to the assumption.

The general claim follows by induction on d. □

In the context of grids, one may use the fact that the maximal degree of a vertex in a grid is 4 along with the previous lemma, and conclude that every vertex $v \in V'$ is equivalent to at most 2 cutset vertices of tree-degree 3. A closer inspection of the previous proof shows that in case a vertex $v \in V'$ is indeed equivalent to 2 cutset vertices c and c' of tree-degree 3, then the induced graph must be of the following general cycle topology (focused on the relevant vertices):

Equipped with the previous observations we are now ready to prove Lemma 20

Proof of Lemma ***20***. **a.** Let $t \in T$ be a tree and let $c \in B(t)$ be a cutset vertex on the boundary of t, such that there does not exist a vertex $c' \in B(t)$, which is higher than c (i.e. with a higher y-coordinate value). Since c is on the boundary of t it touches at least 2 trees, and from the assumption on the tree-degree of the cutset vertices in the graph, it follows that it touches exactly 2 trees, as otherwise it would have tree-degree equal or less than 2. Let $s \in T$ be the second tree touching c. In addition, it should be noted that c does not lie on the edges of the grid, as vertices there have a degree equal or less than 3, and therefore must have an tree-degree of 2 or less if they are boundary vertices. Denote by $nb(c)$ the neighbors of c in G, then for a similar reason it holds that $C \cap nb(c) = \emptyset$,

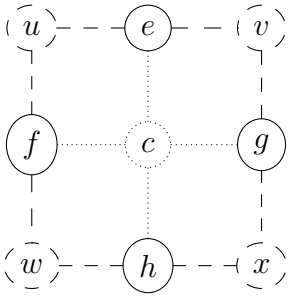


Figure 2.6: Neighborhood of a an extremal cutset vertex.

i.e. no neighbor of c is a cutset node. (refer to diagram 2.6). Assume that vertex f to the left of c belongs to t, then e must belong to t too. This is shown by the following argument: if u is not a cutset vertex then e is connected to f which is in t and thus e is in t. Otherwise, u is a cutset vertex, and assume to the contrary that $e \in s$, then u is a cutset vertex in the boundary of t, which is higher than c, in contradiction to the assumption. It can be shown similarly that if $e \in t$ then $g \in t$, and that if $g \in t$ then $f \in t$. All in all we get that if either one of e, f or g belongs to t than all three belong to t. This implies that if $h \in t$ then $f, g, h \notin t$, since as shown before, if either on of e, f or g belongs to t, then all of them belong to t and along with the assumption that $h \in t$ we get that $e, f, g, h \in t$ in contradiction to the assumption that c is a boundary vertex. It follows that either $|nb(c) \cap V_t| = 1$ and $|nb(c) \cap V_s| = 3$, or the other way around. In both cases, it follows that c has an tree-degree of 3 (either in s in the former case or in t in the latter). Similarly, it can be shown that any vertex $c_2 \in B(t)$, such that there does not exist a boundary vertex $c' \in B(t)$ which is lower than c_2, has an tree-degree of 3.

b. Let $c \in B(t)$ be a vertex touching trees t and s $(t, s \in T)$ with tree-degree 3 in s, then it can be replaced by an equivalent node $v \in V'$. if v is of degree 3 in the graph induced by $V' \cup \{c\}$, then this replacement reduces (by 1) the number of connected components of F. Otherwise, the number of connected components after the replacement remains the same as before. Denote by T' the partition of the forest induced by $(V' \cup \{c\}) \setminus \{v\}$ to trees. If there exist a tree $t' \in T'$ and a vertex $c' \in B(t')$ of tree-degree equal or less than 2, then we return to the previous situation. Otherwise, denote by $nb(v)$ the neighbors of v in the graph induced by $V' \cup \{c, v\}$, and let $t' \in T'$ be the tree such that $|nb(v) \cap V_{t'}| = 1$, then from the preceding claims it follows that there must exist another cutset vertex $v \neq u \in B(t')$ with tree-degree 3 in the graph induced by $(V' \cup \{c\}) \setminus \{v\}$. As before, u can be replaced with a unique vertex $w \in (V' \cup \{c\}) \setminus \{v\}$, and this process continues as long as there does not exist a boundary vertex of tree-degree less than 3. Since the graph is finite and each vertex of tree-degree 3 can be replaced by a unique vertex, this process ought to stop or a vertex x that was previously removed from the cutset will be added to it again. Since at each step the vertex removed is different from the one added in the previous step, the latter condition can only occur if x is equivalent

to 2 cutset vertices q and p. In this case, x may be added to the cutset instead of q, after being previously replaced by p. Let C be the cutset before x was added instead of q, then as can be seen from the diagram in figure 2.5, two of q, p and x must be cutset vertices, and in two of these configurations one of the cutset vertices is of tree-degree 2, we will denote this vertex by y. In figure 2.5, in each of the configurations where v is a cutset vertex, the other cutset vertex has an tree-degree of 2. As seen before, since y has an tree-degree of 2, it is guaranteed that it can be replaced with a vertex with induced degree of 2, thus reducing number of connected components in the induced forest. □

Proof of Theorem 16. Using Lemmas 18 and 20 the main theorem follows immediately: Let G be a grid graph and let C be a cutset such that the induced forest F is disconnected. Assume w.l.o.g. that C does not contain vertices of tree-degree less than 2. If there exists a tree $t \in T$ and a boundary vertex $c \in B(t)$ of tree-degree 2, then Lemma 18 shows that it can be replaced with another vertex while reducing the number of connected components. Otherwise, there does not exist a tree $t \in T$ and a boundary vertex $c \in B(t)$ of tree-degree 2, and therefore Lemma 20 shows that there exists a finite series of replacement moves, such that the forest induced by the resulting cutset contains less connected components than the original forest. □

All in all, we see that for every cutset that induces a forest with more than one connected component a series of replacement moves can be made, which reduces the number of connected components in the induced forest while not adding cutset nodes. As this can be done as long as the induced forest is disconnected, we see by induction that given a cutset C, there exists a cutset C', such that $|C'| \leq |C|$ and the forest induced by C' contains only one connected component, i.e. the induced forest is a single tree. In particular, if the initial cutset C is minimal, then the resulting cutset C' is minimal as well.

2.4 Improved lower bounds

We will use the results of section 2.3 in order to improve the known lower bound on the size of the minimal cutset of $M_{n,m}$. In particular, we will show that our lower bound is equal to the upper bound in these certain cases. In the following we denote by $olb_{n,m}$ the old lower bound of [16], i.e.

$$olb_{n,m} = \left\lceil \frac{(m-1)(n-1)+1}{3} \right\rceil$$

and by $nlb_{n,m}$ the new lower bound obtained by us.

Lemma 22. *Let $G := M_{n,m} = (V, E)$ be the $n \times m$ grid graph, and $C \subseteq V$ a cycle-cutset, such that the graph $T = (V_T, E_T)$ induced by $V' = V \backslash C$ is a single tree. Denote by α the*

number of cutset vertices which lie along the perimeter of the grid but not in its corners, i.e.

$$\alpha = C \cap \left(\left\{(1,j),(n,j) : 1 < j < m\right\} \cup \left\{(i,1),(i,m) : 1 < i < n\right\}\right)$$

*and by β the number of cutset vertices which lie in the corner of the grid, i.e. $\beta = |C \cap \{(1,1),(1,m),(n,1),(n,m)\}|$. Denote by n_C the cardinality of C, and by p the number of connected components of the graph induced by C (**not by** $V \backslash C$). Then it holds that*

$$(n-1)(m-1) + \alpha + 2\beta \leq 2n_C + p \tag{2.1}$$

with equality holding iff the graph induced by the cutset C (and not $V \backslash C$) does not contain cycles.

Proof of Lemma 22. Denote $n_T = |V_T|$. If every vertex of T is of degree 4 in G, then it can be easily shown that the number of edges incident to T from a vertex in C is $2n_T + 2$. Since there are $2(n-2) + 2(m-2) - \alpha$ vertices of T which lie along the boundaries of G, each of which reducing the number of incident edges by 1 (as each of these vertices is only of degree 3 in G), and $4 - \beta$ vertices of T which lie in the corners of G, each of which reducing the number of incident edge by 2, we get that the number of edges A incident to T from C is

$$\begin{aligned} A :&= 2n_T + 2 - [2(n-2) + 2(m-2) - \alpha] - 2(4-\beta) \\ &= 2n_T - 2n - 2m + \alpha + 2\beta + 2 \end{aligned}$$

It can be shown similarly, that if all the connected components of C are trees, then the number of edges B incident to C from T is

$$B := 2n_C + 2p - \alpha - 2\beta$$

and that if not all the connected components of C are trees, then the number of edges incident to C from T is bound from above by B.[1]

Using the facts that $n_T + n_C = n \cdot m$ and that $A \leq B$ one receives

$$2n \cdot m - 2n_C - 2n - 2m + \alpha + 2\beta + 2 \leq 2n_C + 2p - \alpha - 2\beta$$

which after reorganizing gives us the requested inequality:

$$(n-1)(m-1) + \alpha + 2\beta \leq 2n_C + p$$

Where the equality hold if all the connected components of C are trees. □

[1] It should be mentioned that using the results of the previous section, it can be shown that there always exists a cutset, whose connected components are indeed trees (along with all the other aformentioned desired properties).

Table 2.1: Significant functions of r and s

r	s	$(r-1)(s-1)+1$	$(*)$
0	0	2	2
0	1	1	3
0	2	0	1
1	1	1	3
1	2	1	3
2	2	2	2

We note that using the trivial facts that the number p of connected components of C is smaller than $|C| = n_C$, and that $\alpha + 2\beta \geq 1$, as there must be at least one cutset vertex along the perimeter of the grid, one receives from Lemma 22 that it holds that

$$(n-1)(m-1)+1 \leq 3n_C$$

which is a restatement of the lower bound of [16].

Assume that $n \equiv r \mod 3$ and that $m \equiv s \mod 3$ $(0 \leq r, s \leq 2)$, i.e. $n = 3q + r$ and $m = 3p + s$, and assume w.l.o.g that $r \leq s$. Additionally, assume that $n_C = olb_{n,m}$, then by algebraic manipulations it can be shown that

$$\begin{aligned} n_C = olb_{n,m} &= \left\lceil \frac{(m-1)(n-1)+1}{3} \right\rceil \\ &= \left\lceil \frac{9pq + 3pr + 3qs + rs - 3p - s - 3q - r + 2}{3} \right\rceil \\ &= 3pq + p(r-1) + q(s-1) + \left\lceil \frac{(r-1)(s-1)+1}{3} \right\rceil \end{aligned}$$

Referring to table 2.1 we see that the value of the fraction is either $\frac{1}{3}$ and $\frac{2}{3}$, unless $r = 0$ and $s = 2$, and therefore we get that

$$n_C = 3pq + p(r-1) + q(s-1) + \mathbb{1}[r \neq 0 \vee s \neq 2] \tag{2.2}$$

Plugging equation 2.2 in inequality 2.1 we get

$$\begin{aligned} p &\geq (m-1)(n-1) - 2n_C + \alpha + 2\beta \\ &= 3pq + q(s-1) + p(r-1) + (r-1)(s-1) - 2 \cdot \mathbb{1}[r \neq 0 \vee s \neq 2] + \alpha + 2\beta \\ &= n_C + (r-1)(s-1) - 3 \cdot \mathbb{1}[r \neq 0 \vee s \neq 2] + \alpha + 2\beta \end{aligned}$$

Rearranging the expression we get the following inequalities

$$0 \leq n_C - p \leq \underbrace{3 \cdot \mathbb{1}[r \neq 0 \vee s \neq 2] - (r-1)}_{(*)} - \alpha - 2\beta \tag{2.3}$$

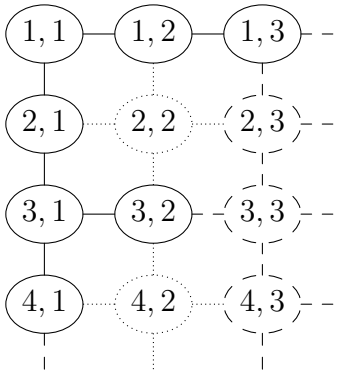


Figure 2.7: The upper-left corner of a grid

$$\alpha + 2\beta \leq \underbrace{3 \cdot \mathbb{1}\left[r \neq 0 \vee s \neq 2\right] - (r-1)}_{(*)} \tag{2.4}$$

These inequalities are at the core of proving the improvements to the lower bounds of [16].

Theorem 23. *Let $m, n \geq 4$, such that $n \equiv 0 \mod 3$ and $m = 2 \mod 3$, and assume that at least one of n and m is even, then the size of the minimal cutset of the $n \times m$ grid $M_{n,m}$ (or the $m \times n$ grid $M_{m.n}$) is at least $olb_{n,m} + 1$, i.e. $nlb_{n,m} = olb_{n,m} + 1$.*

Proof of Theorem 23 . As stated by inequality 2.4 and table 2.1, we can see that $\alpha + 2\beta \leq 1$, which implies that $\alpha = 1$ and $\beta = 0$, i.e. there exists a single cutset vertex along the boundaries of the grid, not including the corners. Assume w.l.o.g that it is located along the right boundary of the grid. Focusing on the 2×2 sub-grids containing each of the four corners of the grid, we note that since $m, n \geq 4$ all four sub-grids are disjoint, and since there exists only a single cutset vertex along the boundaries, in at least three of the four 2×2 sub-grids three of the vertices cannot be cutset vertices, and therefore the forth - inner - vertex must be a cutset vertex, in order to open the cycle formed by the 2×2 sub-grid (refer to Figure 2.7, depicting the upper-left corner of the grid). Assume w.l.o.g that $v_{2,2}$, $v_{2,m-1}$, $v_{n-1,2}$ are cutset vertices, and assume that w.l.o.g that n is even. Following from inequality 2.3 we see that the number of connected components of the cutset is equal to the number of cutset vertices, that is C is an independent set. Since $v_{2,2}$ is a cutset vertex, it follows that $v_{4,2}$ must be a cutset vertex too, in order to break the cycle formed by $v_{3,1}$, $v_{3,2}$, $v_{4,2}$ and $v_{4,1}$. For a similar reason, $v_{6,2}$ must be in the cutset, and so on and so forth: for every even number i, $v_{i,2}$ must be a cutset vertex. Since n is even by assumption, we get that $n - 2$ is even and therefore $v_{n-2,2}$ is a cutset vertex. Remembering that $v_{n-1,2}$ is a cutset vertex we get a contradiction to the fact that C is an independent set. □

Let m, n be two integers at least one of which is even. Assume w.l.o.g. that m is even and that $m \geq 6$. Using the upper bounds $ub_{n,m}$ of [17], we get that $nlb_{n,m} = ub_{n,m}$, if

$n \geq 9$ and $n \equiv 0 \mod 3$ or if $n \geq 11$ and $n \equiv 2 \mod 3$, i.e. in every case in which [17] have shown an upper bound applicable in the conditions of Theorem 23, the upper bound is equal to the lower bound.

Theorem 24. *Let $m, n \geq 4$, such that both $n \equiv 0 \mod 3$ and $m = 0 \mod 3$ or both $n \equiv 2 \mod 3$ and $m = 2 \mod 3$, and assume that both n and m are even, then the size of the minimal cutset of the $n \times m$ grid $M_{n,m}$ (or the $m \times n$ grid $M_{m.n}$) is at least of size $olb_{n,m} + 1$, i.e. $nlb_{n,m} = olb_{n,m} + 1$.*

Before proving this theorem we would need to introduce some additional definitions and lemmas which will be used in the proof.

Definition 25. Let $v_{i,j}$ be a vertex in the grid graph $M_{n,m}$, we say that it is an *even vertex* if $i + j$ is even, and that it is an *odd vertex* if $i + j$ is odd. Let V_e be the set of all even vertices in $M_{n,m}$, and define $E_e = \{(v_{i,j}, v_{k,l}) \in V_e^2 : |i - k| = 1, |j - l| = 1\}$. We call the graph $G_e = (V_e, E_e)$ *the even semi-grid*, and call adjacent vertices in the semi-grid *semi-neighbors*. The *odd semi-grid* $G_o = (V_o, E_o)$ is defined similarly over the set of odd vertices V_o.

Lemma 26. *Let $G = (V, E)$ be a grid graph, and $A \subseteq V$ an independent set in G, and let S be a connected component of the graph induced by $A \cap V_e$ ($A \cap V_o$) in the even (odd) semi-grid, then:*

a. *if no vertex in S lies on the boundaries of G, then there exists a cycle in the graph induced by $V \backslash A$ in G.*

b. *if there exists a cycle in S (with edges in E_e (E_o)), then S separates G to (at least) two connected components, i.e. the graph induced by $V \backslash S$ in G contains at least two connected components.*

c. *if there exist two vertices in S on the boundaries of G, then S separates G to (at least) two connected components.*

Proof. **a.** The proof of the lemma is by induction on the number of vertices in S.

b. The proof follows from the fact that a cycle in G_o separates the plane to two (non-empty) regions.

c. The proof follows by connecting the two vertices on the boundaries of S by a new edge in G_o (E_e) and using previous claim. □

Definition 27 (stem)**.** Let $G = (V, E)$ be a grid graph, and $A \subseteq V$ an independet set, and let S be a connected component of the graph induced by $A \cap V_e$ ($A \cap V_o$) in the even (odd) semi-grid, we call the vertices of S along the boundaries of the grid *the stems of S*.

Definition 28 (even/odd semi-tree)**.** Let $G = (V, E)$ be a grid graph, and $A \subseteq V$ an independent set, and let S be a connected component of the graph induced by $A \cap V_e$ ($A \cap V_o$) in the even (odd) semi-grid. If S does not contain a cycle, then we call S *an even (odd) semi-tree*.

Corollary 29. *Let G be a grid, and C an independent set, such that the graph induced by $V\backslash C$ on G is a tree. Then, following from lemma 26 is that C can be partitioned to a set of disjoint single-stemmed semi-trees.*

Proof of Corollary 29. Let S be a connected component of the graph induced by $A \cap V_e$ on the even semi-rind. By lemma 26a, the fact that the graph induced by $V\backslash C$ on G is acyclic implies that there exists a vertex in S that lies on the boundaries of the grid. By lemma 26c, the fact that graph induced by $V\backslash C$ is connected implies that the there exists exactly one vertex in S which lies on the boundary of G, and there for S is single stemmed. Finally, by lemma b, the fact that graph induced by $V\backslash C$ is connected implies that there are no cycles in S and therefore it is a semi-tree. □

Proof of Theorm 24. We see from Inequality 2.4 and Table 2.1 that in both these cases $\alpha + 2\beta \leq 2$, which may result in several scenarios:

1. If $\alpha = 0$ and $\beta = 1$, then we get from Inequality 2.3 that the cutset is an independent set and therefore using a claim similar to that of Theorem 23 we receive a contradiction.

2. If $\alpha = 1$ and $\beta = 0$, then we get from Inequality 2.3 that all the connected components of the cutset are singletons apart from one which contains two vertices. Following the lines of the proof for Theorem 23, we note that at least in three of 2×2 grids located the corners of the $M_{n,m}$ the inner vertex must be a cutset vertex. Assume w.l.o.g that the cutset vertex lies along the bottom edge of the grid, and that in all 2×2 grids in the corners except maybe that in the lower-right one the inner vertex is a cutset vertex. As before, every second vertex along the second row and along the second column of the graph must be cutset vertices, giving us two pairs of adjacent cutset vertices - in the upper-right corner and in the lower-left corner, in contradiction to the fact that only one such pair should exist.

3. If $\alpha = 2$ and $\beta = 0$, then again we get that the cutset is an independent set. If at least one of the cutset vertices along the boundaries of the grid does not lie in a 2×2 grid in a corner of the graph, then there are at least three 2×2 grids in the corners, in which the inner vertex is a cutset vertex. Since there are two pairs of 2×2 corner sub-grids sharing along the same boundary such that their inner vertex is a cutset vertex and only a single vertex that is on a boundary of the grid, we get that there exists a boundary such that in both its corners the inner vertex of the 2×2 grid is a cutset vertex, and there is no cutset vertex along it. Consequently, it follows as before, that every second vertex along the edge is a cutset vertex. Therefore, we get that there are two adjacent cutset vertices, in contradiction to the fact the the cutset is an independent set.

Therefore, assume that both cutset along the boundaries of the grid, lie in the 2×2 grid of adjacent corners A and B, then in the 2×2 grid of the two remaining corners C and D, the inner vertex is a cutset vertex. Since there is no cutset vertex along the boundary between C and D, we get that there exist two adjacent cutset vertices, similarly to before - a contradiction.

Finally, assume that the cutset vertices along the boundaries of the grid lie in the 2×2 grid of opposite corners. Assume w.l.o.g that they lie in the 2×2 grids of the upper-

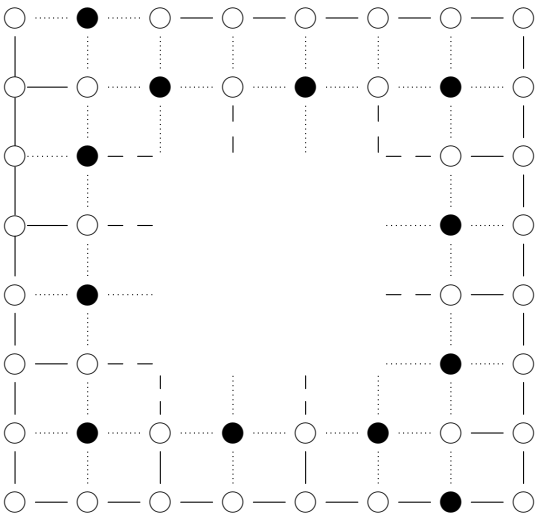


Figure 2.8: A possible frame of a 8×8 grid with a cutset of size $lb_{8,8}$. Cutset vertices are marked with a black circle and tree vertices are marked with white circle. Other notation as before.

left and lower-right corners (refer to Figure 2.8. Note that in this case both boundary vertices are necessarily odd. As they are the only possible stems for the cutset, it follows from corollary 29 that all cutset vertices in the graph are odd as well. Additionally, as before we get that $v_{i,2}, v_{2,i}, v_{j,m-1}, v_{n-1,j} \in C$ for every odd i and even j. Note that the frame of the grid is separated to two parts by the cutset vertices. Since the the graph induced by $V \backslash C$ is a connected tree by assumption, we get that there must be a path from one part to the other though the inner part of the grid. Assume w.l.o.g that the path begins at $v_{1,j}$ for a (necessarily) even j ($4 \leq j \leq m-2$) and goes though $v_{2,j}$ to $v_{3,j}$. Since $v_{3,j-1}, v_{2,j+1}, v_{3,j} \notin C$ since they are even, we get that $v_{4,j-1}, v_{4,j+1} \in C$ (as otherwise two cycles would form). Since all four semi-neighbors of $v_{3,j}$ are cutset vertices, we get that the path must continue 2 steps at a time in any direction, because the cutset is an independent set. In general, assume the path reaches vertex $v_{k,l}$ such that $k \equiv 1 \mod 2$ and $l \equiv j \mod 2 \equiv 0 \mod 2$, then since $v_{k-1,l}$, $v_{k,l-1}$, $v_{k+1,l}$, $v_{k,l+1}$ are all even and therefore not cutset vertices, we get that all four semi-neighbors of $v_{k,l}$ must be cutset vertices, and the path must continue 2 steps in any direction. As a result, we get that the path must be connected to the other part of the grid's frame though a vertex of either the form $v_{k,2}$ with $k \equiv 1 \mod 2$, i.e. an odd k, or the form $v_{n-1,l}$ with $l \equiv 0 \mod 2$, i.e. an even l. However, as seen before, all vertices of such forms are necessarily cutset vertices, and therefore the path could not pass though them - a contradiction.

□

Let m, n be two even integers, and assume that $m \geq 6$. If $n \geq 9$ and $n \equiv 0 \mod 3$, then again case (iii) in [17] shows that the upper bound on the size of the minimal cutset

Table 2.2: Summery of known bounds on the size of a minimal cycle-cutset in big grids

n (mod 3)	m (mod 3)	m/n even	known upper bound
1	-	-	$olb_{n,m}$
0	0	at most one	$olb_{n,m}+1$
	2	neither	$olb_{n,m}+2$
2	2	at most one	$olb_{n,m}+2$
0	0	both	$olb_{n,m}+1=nlb_{n,m}$
	2	at least one	$olb_{n,m}+1=nlb_{n,m}$
2	2	both	$olb_{n,m}+1=nlb_{n,m}$

is $ub_{n,m} = olb_{n,m} + 1 = nlb_{n,m}$. If $n \geq 11$ and $n \equiv 2 \mod 3$, in case (iv) in [17] shows that $ub_{n,m} = olb_{n,m} + 1 = nlb_{n,m}$. To conclude, in every case in which [17] have shown an upper bound applicable in the conditions of Theorem 24, the upper bound is equal to the lower bound.

Let m, n be two even integers, and assume that $m \geq 6$. Using the upper bounds $ub_{n,m}$ of [17], we get that $nlb_{n,m} = ub_{n,m}$, if $n \geq 9$ and $n \equiv 0 \mod 3$ or if $n \geq 11$ and $n \equiv 2 \mod 3$, i.e. in every case in which [17] have shown an upper bound applicable under the conditions of Theorem 24, the upper bound is equal to the lower bound.

2.5 Conclusion

We have established the basis to the notion of tree-inducing cycle-cutsets and the transformation of a general cutset to such a cutset. We have shown that in grids one can always transform a cycle-cutset to a tree-inducing cutset with no more vertices than the original one. These results lay the foundation to a more elaborate method of analyzing and bounding the size of minimal cutsets, thus allowing us to improve its lower bound in some cases. In other cases, a gap between the lower and the upper bounds remains, and more meticulous analysis should be undertaken in order to characterize better the classes in which the lower bound can be raised.

We summarize the relation between the previously known upper bounds on the size of a minimal cycle-cutset and the lower bound in grids $M_{n,m}$ where $n, m \geq 11$ along with our new results in table 2.2, which clearly displays the current gaps between the lower and the upper bound.

Bibliography

[1] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, et al. Interactive Digital Photomontage. *ACM Trans. Graph.*, 23(3):294–302, August 2004.

[2] Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized Algorithms for the Loop Cutset Problem. *J. Artif. Int. Res.*, 12(1):219–234, May 2000.

[3] Umberto Bertele and Francesco Brioschi. *Nonserial Dynamic Programming*. Academic Press, Inc., Orlando, FL, USA, 1972.

[4] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.

[5] Simon de Givry, Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Solving Max-SAT as Weighted CSP. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 363–376. Springer Berlin Heidelberg, 2003.

[6] Rina Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artif. Intell.*, 41(3):273–312, January 1990.

[7] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(12):41 – 85, 1999.

[8] Rina Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[9] Rina Dechter and Irina Rish. Mini-buckets: A General Scheme for Bounded Inference. *J. ACM*, 50(2):107–153, March 2003.

[10] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, April 1982.

[11] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Efficient Stochastic Local Search for MPE Solving. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 169–174, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[12] Richard M. Karp. Reducibility among Combinatorial Problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[13] Kalev Kask and Rina Dechter. A Graph-based Method for Improving GSAT. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, pages 350–355. AAAI Press, 1996.

[14] Kalev Kask and Rina Dechter. Stochastic local search for bayesian networks. In *In Workshop on AI and Statistics*, pages 113–122, 1999.

[15] Junkyu Lee, William Lam, and Rina Dechter. Benchmark on DAOOPT and GUROBI with the PASCAL2 Inference Challenge Problems. In *DISCML*, 2013.

[16] Flaminia L. Luccio. Almost Exact Minimum Feedback Vertex Set in Meshes and Butterflies. *Inf. Process. Lett.*, 66(2):59–64, April 1998.

[17] Florent R. Madelaine and Iain A. Stewart. Improved upper and lower bounds on the feedback vertex numbers of grids and butterflies. *Discrete Mathematics*, 308(18):4144 – 4164, 2008.

[18] Radu Marinescu and Rina Dechter. AND/OR Branch-and-Bound Search for Combinatorial Optimization in Graphical Models. *Artif. Intell.*, 173(16-17):1457–1491, November 2009.

[19] Radu Marinescu, Kalev Kask, and Rina Dechter. Systematic vs. Non-systematic Algorithms for Solving the MPE Task. In *UAI-03*, 2003.

[20] Patrick Mills and Edward Tsang. Guided local search for solving SAT and weighted MAX-SAT problems. In *Journal of Automated Reasoning*, pages 89–106. IOS Press, 2000.

[21] James D. Park. Using Weighted MAX-SAT Engines to Solve MPE. In *Eighteenth National Conference on Artificial Intelligence*, pages 682–687, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[22] Judea Pearl. Reverend Bayes on inference engines: a distributed hierarchical approach. In *in Proceedings of the National Conference on Artificial Intelligence*, pages 133–136, 1982.

[23] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[24] Gadi Pinkas and Rina Dechter. Improving Connectionist Energy Minimization. *J. Artif. Int. Res.*, 3(1):223–248, October 1995.

[25] Richard Szeliski, Ramin Zabih, Daniel Scharstein, et al. A comparative study of energy minimization methods for Markov random fields. In *In ECCV*, pages 16–29, 2006.

[26] Olga Veksler. *Efficient Graph-Based Energy Minimization Methods In Computer Vision*. PhD thesis, 1999.

[27] Christos Voudouris. *Guided local search for combinatorial optimisation problems*. PhD thesis, Department of Computer Science, University of Essex, 1997.

[28] Benjamin W. Wah and Yi Shang. Discrete Lagrangian-Based Search for Solving MAX-SAT Problems. In *IJCAI (1)*, pages 378–383, 1997.

Appendix A

Result Plots

Following are more detailed graphs presenting the evolution of the algorithms compared in Section 1.9 over time on specific problems. Each figure presents the behavior of the algorithms on a specific problem set and each plot presents the results obtained after a certain time limit specified in the plots title. For every problem instance a marker denotes the average normalized result (over 10 independent runs) obtained by each of the algorithms after the specified time bound. Bars above and below the marker denote the minimum and the maximum normalized results (over the 10 runs) obtained by each algorithm. Markers beneath 0 indicate that the corresponding algorithm has failed to return any value within 1.5 of the matching time bound due to long initialization. If the markers of all algorithms are below 0 (in some CSP problems), than no improvement was observed during the entire experiment, i.e. the optimal solution was found within less than 0.1 seconds.

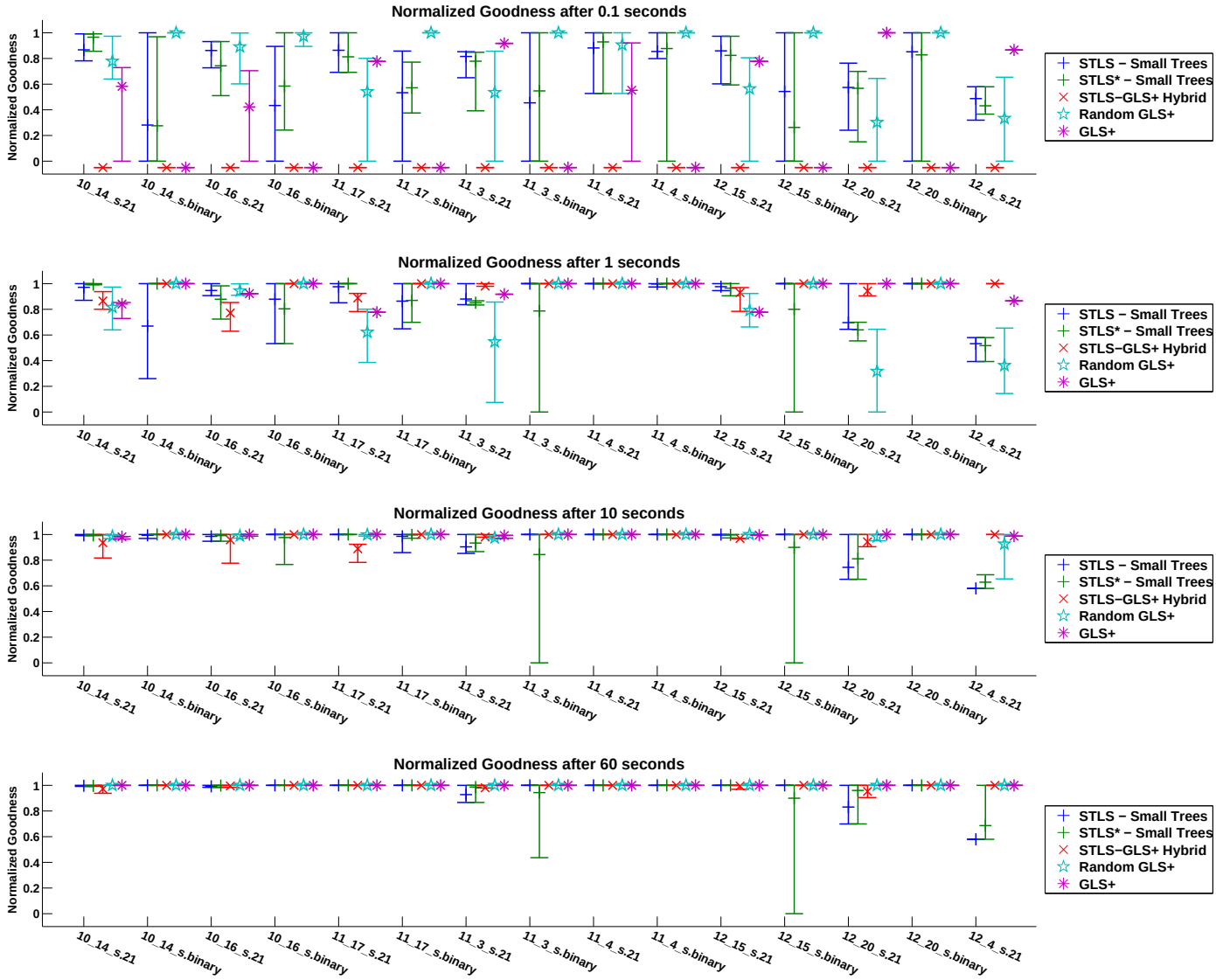


Figure A.1: Results on some problems from the Segmentation domain. For each problem the resulting energies were linearly normalized such that the highest energy obtained by some algorithm at some time bound corresponds to 1 and the lowest corresponds to 0. Markers beneath 0 indicate that the corresponding algorithm has failed to return any value within 1.5 of the matching time bound due to long initialization. The average normalized energy obtained by each of the algorithms is denoted by a marker according to the algorithm, and the minimal and maximal normalized energies are depicted as vertical lines. "STLS - Small Trees" is the basic version of BPLS (named "BPLS - Basic" in chapter 1), "STLS* - Small Trees" is the basic version of BPLS*, i.e. BPLS - Basic with the heuristics presented in section 1.7, STLS-GLS+ Hybrid" is the hybrid of BPLS - Basic and GLS$^+$ initialized with the mini-bucket heuristic, "Random GLS+" is GLS$^+$ initialized randomly and "GLS+" is GLS$^+$ initialized with the mini-bucket heuristic. Note that the figure displays the results at different time bounds.

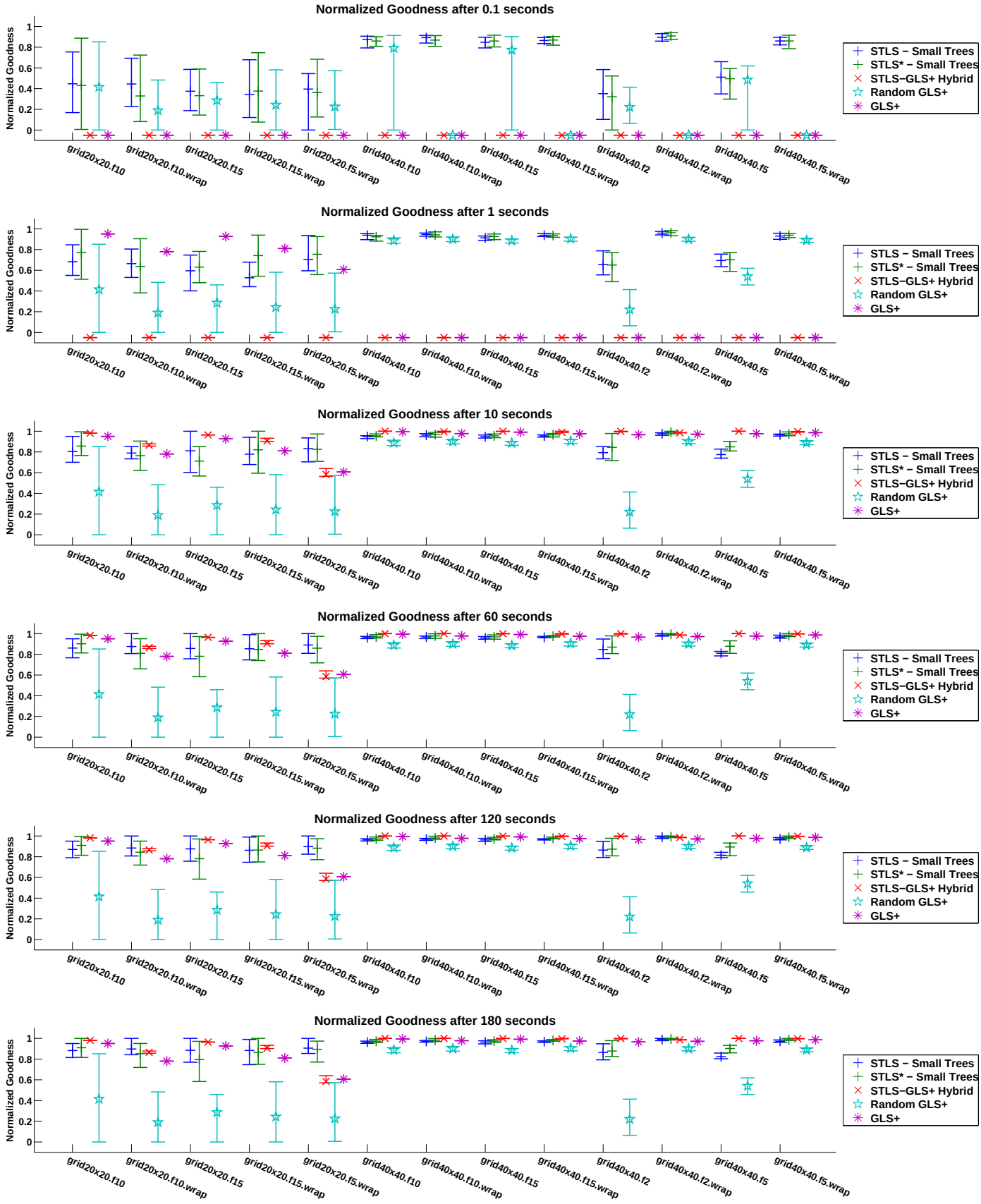


Figure A.2: Results on 20×20 and 40×40 grids from the Grids domain. See Figure A.1 for details.

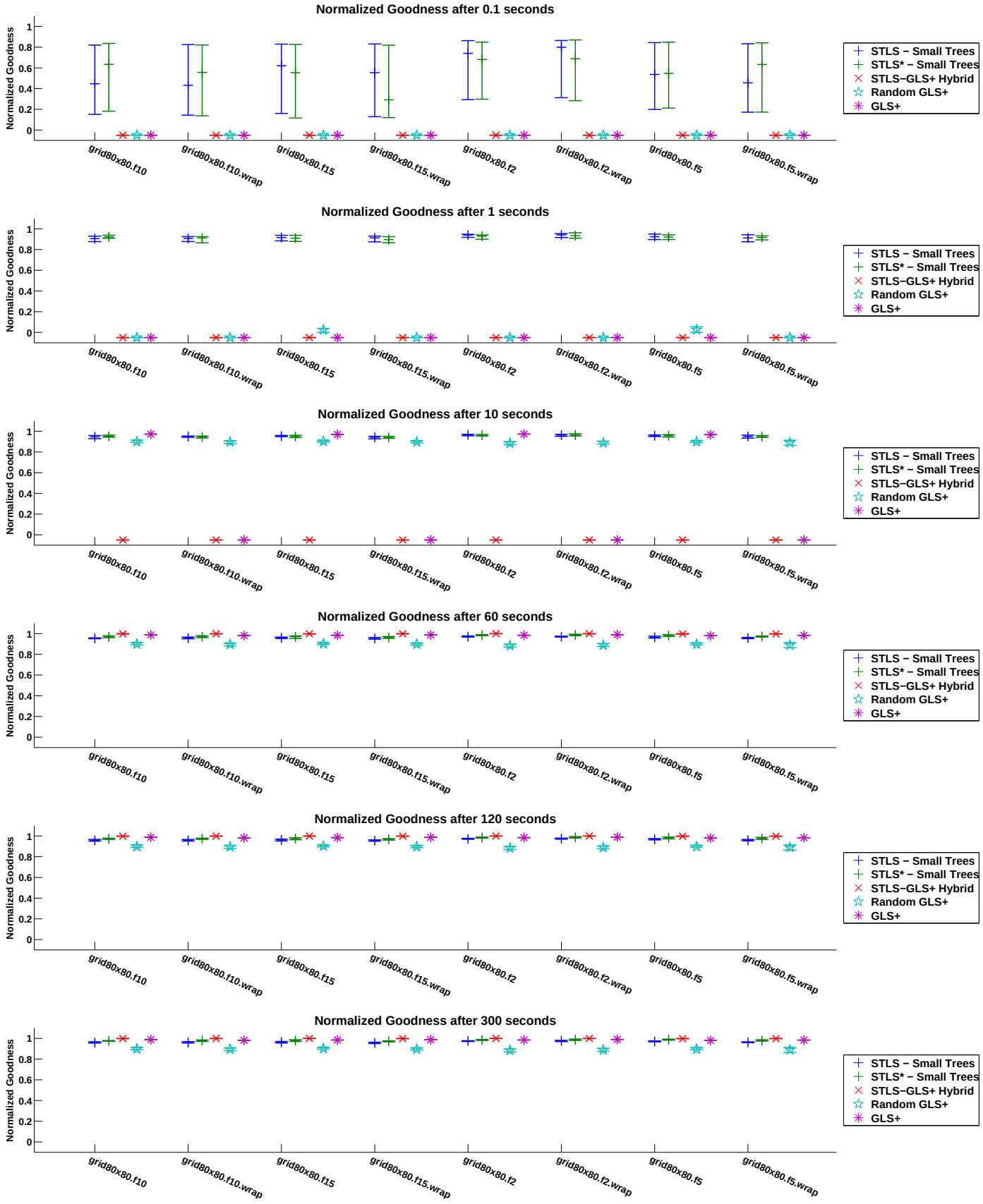


Figure A.3: Results on 80 × 80 grids from from the Grids domain. See Figure A.1 for details.

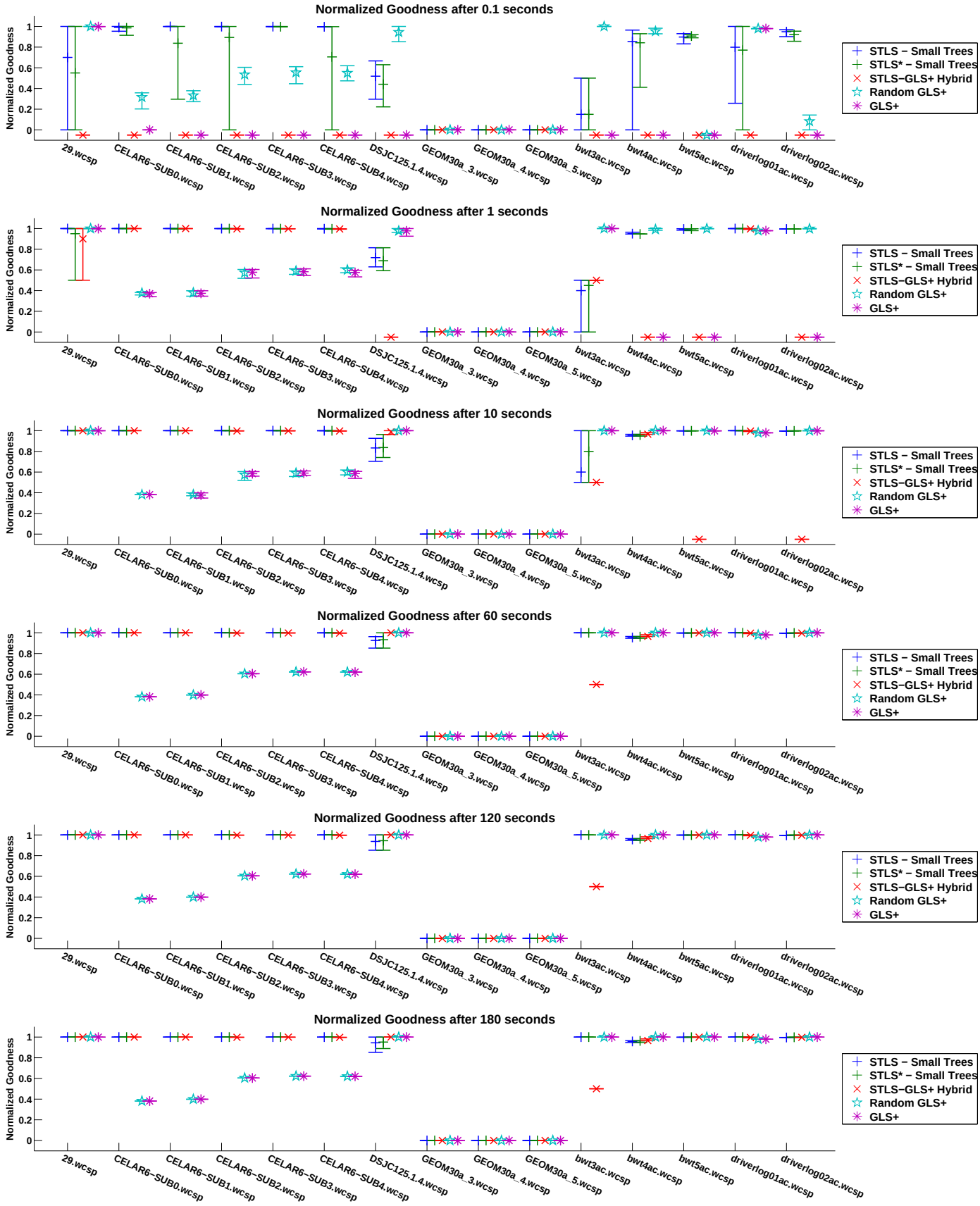


Figure A.4: Results on the CSP domain. See Figure A.1 for details.

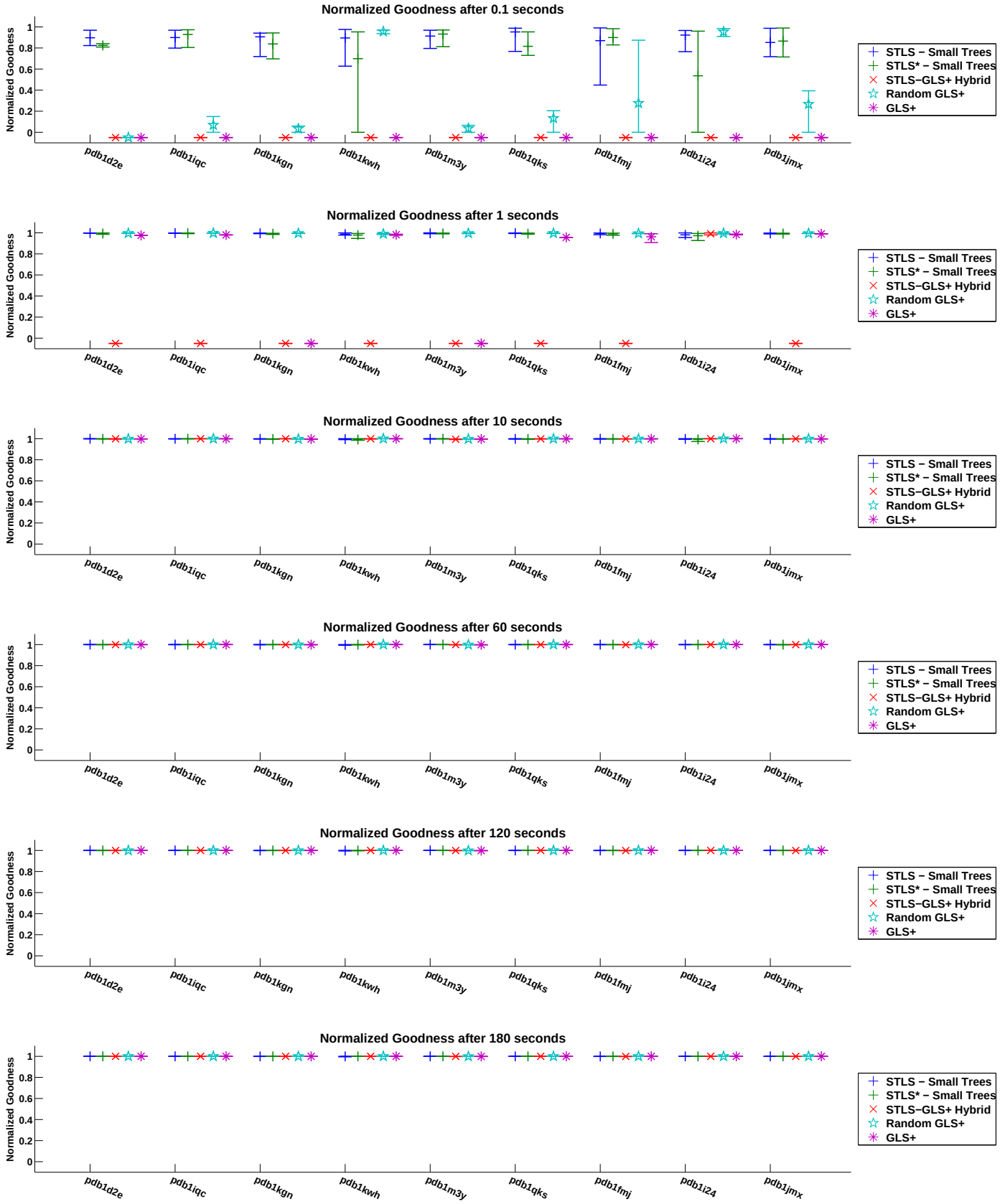


Figure A.5: Results on the problems from the Protein Folding domain. See Figure A.1 for details.